



AccuRev®

CLI User's Guide

Version 7.1

Revised 30-October-2017

Copyright and Trademarks

Copyright © Micro Focus 2017. All rights reserved.

This product incorporates technology that may be covered by one or more of the following patents:

U.S. Patent Numbers: 7,437,722; 7,614,038; 8,341,590; 8,473,893; 8,548,967.

AccuRev, **AgileCycle**, and **TimeSafe** are registered trademarks of Micro Focus.

AccuBridge, **AccuReplica**, **AccuSync**, **AccuWork**, **Kando**, and **StreamBrowser** are trademarks of Micro Focus.

All other trade names, trademarks, and service marks used in this document are the property of their respective owners.

Table of Contents

Preface	vii
Using This Book	vii
Typographical Conventions	vii
Contacting Technical Support.....	vii
1. Overview of the AccuRev®	
Command-Line Interface	1
Using AccuRev with a Secure AccuRev Server.....	1
Working with Files in a Workspace	3
Working with Multiple Repositories	4
File Locking in Workspaces	5
Selecting Elements Based on their Status.....	10
Performance Considerations for Element-Selection Commands	12
Selecting Elements Using Filename Patterns	13
Combining File-Status Filters with Filename Patterns.....	14
Selecting Elements Using Element-IDs	14
Creating a Workspace.....	15
Placing Files Under Version Control.....	17
Editing Files in a Workspace.....	17
Checkpointing — Saving Private Versions	17
Comparing Versions of a Text File	18
Making Your Changes Public.....	18
Concurrent Development — Working Well with Others.....	19
Determining the Status of Files	19
Getting Element History	20
Tracking Other Users’ Work	20
Incorporating Other Users’ Work into Your Workspace	21
Concurrent Development — When Streams Collide	21
Managing a Depot’s Stream Hierarchy	22
Managing and Creating New Versions of Files.....	24
Getting Status Information	26
Include/Exclude Rule Facility	27
Administration	27
Managing Users and Security.....	28
Managing Change Packages.....	29
2. AccuRev® Command-Line	
Reference	31
Command Summary.....	31
add.....	36
addmember	40
anc	41
anchor.....	45

annotate	47
archive	49
authmethod	54
backup	55
cat	56
chdepot	58
chgroup	60
chmod	61
chpasswd	62
chref	63
chslice	65
chstream	67
chuser	69
chws	71
clear	73
co	74
cpkadd	76
cpkdepend	78
cpkdescribe	82
cpkremove	84
defunct	86
demote	90
diag	94
diff	95
disable_ssl	101
eacl	102
enable_ssl	105
excl	107
files	109
get_certificate	112
getconfig	114
getpref	116
getproperty	117
help	119
hist	120
hostinfo	126
incl	127
incldo	129
info	131
ismember	133
issuediff	134
issuelist	135
keep	137
licenses	139
ln	141
lock	144
login	146
logout	148
lsacl	149
lsrules	150

merge.....	152
mergelist.....	159
mkdepot.....	162
mkgroup.....	164
mkref.....	165
mkreplica.....	167
mkrules.....	168
mksnap.....	169
mkstream.....	171
mktrig.....	175
mkuser.....	178
mkws.....	180
move.....	184
name.....	186
patch.....	188
patchlist.....	191
pop.....	193
promote.....	197
purge.....	203
putconfig.....	206
reactivate.....	207
rebase.....	208
reclaim.....	210
remove.....	211
replica.....	212
revert.....	214
rmmember.....	217
rmproperty.....	218
rmreplica.....	219
rmtrig.....	220
rmws.....	221
secinfo.....	222
setacl.....	223
setpref.....	226
setproperty.....	228
show.....	232
sitehist.....	238
start.....	242
stat.....	243
synctime.....	251
touch.....	252
translist.....	253
unarchive.....	254
undefunct.....	256
unlock.....	258
update.....	259
upgrade_client.....	263
verinfo.....	264
wip.....	265
xml.....	267

ENV_VARS..... 268

3. AccuWork™ Command-Line

Interface..... 271

Overview 271

- Determining the Field-ID / Field-Name Correspondence 273
- Using the replaceAll= Attribute 279
- Bulk-updating Issue Records..... 280
- Using 'modifyIssue' to Create New Issue Records..... 280
- Adding Entries to a Change Package 281
- Removing Entries from a Change Package 282
- Listing the Contents of a Change Package 282
- Listing Transactions that Affected Change Packages 283
- Creating a Relationship between Two Issue Records 283
- Removing a Relationship between Two Issue Records..... 283
- Listing Issue Record Relationships 283
- Determining Which Change Packages Are Included in a Stream 284
- Comparing Two Streams in Terms of Change Packages 285

4. XML Output Reference..... 287

How Command Information is Organized..... 287

- Additional Information 288

Preface

This book describes the AccuRev command-line interface (CLI) and how to use it to accomplish everyday AccuRev tasks like creating workspaces, keeping and promoting work, and diffing files and issues.

Using This Book

This book assumes you are familiar with general AccuRev concepts and behavior. Book contents are summarized in the following table:

Chapter	Description
Chapter 1 Overview of the AccuRev® Command-Line Interface	Provides an orientation to the AccuRev command-line interface.
Chapter 2 AccuRev® Command-Line Reference	Provides a detailed description of the accurev program, the main command-line tool in the AccuRev configuration management system.
Chapter 3 AccuWork™ Command-Line Interface	Describes aspects of the command-line interface relevant to the AccuWork issue management system.
Chapter 4 XML Output Reference	Provides detailed information on the XML output of certain CLI commands.

Typographical Conventions

This book uses the following typographical conventions:

Convention	Description
<code>blue sans-serif</code>	Used for sample code or output.
<code>red monospace</code>	Used for examples.
bold	Used for command names, and button names in the AccuRev user interface
<i>light italic</i>	Used for emphasis, book titles, and for first use of important terms
<i>blue italic</i>	Identifies a hyperlink (to a page or Web URL, for example)

Contacting Technical Support

Micro Focus offers a variety of options to meet your technical support needs as summarized in the following table.

For	Visit
Information about technical support services	http://supportline.microfocus.com/
Information about platforms support	http://supportline.microfocus.com/prodavail.aspx

For	Visit
Product downloads and installations	http://supportline.microfocus.com/websync/productupdatesearch.aspx
Product documentation	http://supportline.microfocus.com/productdoc.aspx
SupportLine phone numbers, listed by country	http://www.microfocus.com/about/contact/support/assistance.aspx

When you contact Micro Focus technical support, please include the following information:

- The version of AccuRev and any other AccuRev products you are using (AccuSync or GitCentric, for example).
- Your operating system.
- The version of relevant third-party software (if you are using AccuSync, for example, the version of your ITS).
- A brief description of the problem you are experiencing. Be sure to include which AccuRev interface you were using (Web user interface, Java GUI, or CLI), any error messages you received, what you were doing when the error occurred, whether the problem is reproducible, and so on.
- A description of any attempts you have made to resolve the issue.
- A simple assessment of how the issue affects your organization.

1. Overview of the AccuRev® Command-Line Interface

This chapter provides an orientation to the AccuRev command-line interface (CLI). Before continuing, make sure that:

- You have installed the AccuRev client software on your computer.
- The AccuRev server and database server software has been installed on a computer to which you have a network connection. Running the client and server software on the same machine is fine, too; this is typical for a pre-sales evaluation of the product.
- You have read the *AccuRev Concepts Manual*.

The ‘accurev’ Program

The AccuRev command-line interface is implemented by a program named *accurev*. You can use this tool in a command shell (UNIX/Linux) or at a DOS prompt (Windows). You can also invoke this tool as part of a shell script or batch file, or from a scripting language such as Perl.

Each invocation of the *accurev* program looks like this:

```
accurev <command-name> <options-and-arguments>
```

After executing the specified command, *accurev* returns control to the command shell. (There is no way to execute several AccuRev commands in a single invocation of *accurev*.)

Using AccuRev with a Secure AccuRev Server

AccuRev supports the Secure Sockets Layer (SSL) protocol to provide encrypted communication between the AccuRev client and the AccuRev server. Setup and configuration of SSL is managed by your AccuRev administrator. This section describes the situations in which you, as the end-user, are likely to experience SSL if it has been enabled on your AccuRev server.

The first time you start AccuRev, whether through the GUI or the command line, AccuRev will prompt you to enable SSL for that client. This process allows you to inspect the SSL certificate, and either accept or reject it. If you accept it, AccuRev downloads the certificate to the client machine, enabling you to connect to the AccuRev server. If you reject it, you cannot connect to the AccuRev server. This will prevent you from starting the AccuRev GUI or executing any AccuRev commands.

Once you accept the SSL certificate for your AccuRev client, you are prompted to take action in the future only if:

- **The SSL certificate expires or has been changed.** If your original certificate expires, or if it is no longer recognized by the AccuRev client, AccuRev prompts you to get a new certificate the next time you start AccuRev or execute an AccuRev command. In this case, you again have the opportunity to inspect the new certificate. If you accept it, AccuRev replaces the current certificate with a new one and you can continue using AccuRev. If you reject it, you will not be able to connect to the AccuRev server.

- **SSL is disabled on the AccuRev server.** If the administrator disables SSL on your AccuRev server after you have already accepted a certificate, the next time you start AccuRev or execute an AccuRev command you are informed of this situation and given the choice to disable SSL on your AccuRev client. If you choose to disable SSL, you will be able to continue using your AccuRev client, but the client-server communication will no longer be encrypted.

Note: AccuRev recommends that before disabling SSL on your client you bring this situation to the attention of your AccuRev administrator.

Workspaces and Depots

An AccuRev workspace is a directory hierarchy on your hard disk, containing files that are being managed by AccuRev. You can create any number of workspaces — different ones for different development projects.

Note: Usually, you can think of a workspace as simply containing a collection of files. Sometimes, though, it helps to adopt AccuRev's viewpoint: each file is a version-controlled element; your workspace contains a copy of a certain version of that element.

The AccuRev repository is organized into a set of storage depots. Each depot has a directory hierarchy for physical file storage (containing a set of version-controlled files and directories), and a separate schema in the database for metadata storage. Depots cannot overlap and cannot be contained within one another.

Each workspace corresponds to a particular depot. If you need to work with the files in three different depots, you'll need three different workspaces. Like the depots, the workspaces cannot overlap or be contained within one another.

Example: suppose a depot is structured like this:

```
src
  Red.java
  White.java
  Blue.java
test
  setup.pl
  testrun.pl
  analyze.pl
doc
  Colors.doc
```

Any number of users can create workspaces for themselves, to work with the files in this depot. On a Windows system, you might create a workspace at location **C:\dvt_work**. The depot's subdirectories would be located in your workspace at:

```
C:\dvt_work\src
C:\dvt_work\test
C:\dvt_work\doc
```

You can create a workspace at any location you wish (as long as you have permission to access the disk storage). And you can move an existing workspace to another location — say, to a network drive with a

larger capacity and a regular backup regimen. If you moved the workspace above to **H:\Projects\ColorWheel\derek**, then the three subdirectories would now be located at:

```
H:\Projects\ColorWheel\derek\src
H:\Projects\ColorWheel\derek\test
H:\Projects\ColorWheel\derek\doc
```

AccuRev needs to keep track of a workspace's location. So you specify a pathname when you create the workspace:

```
accurev mkws ... -l C:\dvt_work ...
```

And if you move the workspace, using ordinary operating system tools, you must inform AccuRev of the new location:

```
accurev chws ... -l H:\Projects\ColorWheel\derek ...
```

Working with Files in a Workspace

Your workspace is a private work area. The files in your workspace belong to you. You can edit them whenever you like — no special “check out” command is required. For example, to edit the **Colors.doc** document, you might start a word processor and specify the pathname

H:\Projects\ColorWheel\derek\doc\Colors.doc.

When you specify files to an *accurev* command, however, you usually don't use full pathnames. In general, you *cd* (“change directory”) to the workspace, then use relative pathnames to refer to files when issuing *accurev* commands.

Alternatively, you can specify a location within a workspace to the *accurev* command using the *-L* option, which is available from:

```
accurev stat -x -L \Projects\Colorwheel\derek\doc
```

As usual, you can use simple filenames for files in the current directory:

```
accurev stat Colors.doc
```

And you can use relative pathnames for files in other directories in the same workspace:

```
accurev stat ..\test\setup.pl
```

You can even use absolute (full) pathnames to specify files:

```
accurev stat C:\Projects\Colorwheel\derek\doc
```

You can mix different ways of specifying files on the command line:

```
accurev keep -c "bug fix" colors.doc ..\setup.pl
```

But you cannot specify files from different workspaces in the same *accurev* command.

Depot-Relative Pathnames

In addition, the *accurev* program recognizes a special kind of pathname, termed a depot-relative pathname. This is the pathname to an element from the top-level directory of the depot — or equivalently, from the top-level directory of the workspace. Examples:

```
\\.src\Blue.java
\\.test\analyze.pl
```

The special prefix — *\\.* for Windows, */./* for UNIX/Linux — that distinguishes a depot-relative pathname from an ordinary relative pathname. If your current directory is anywhere within a workspace, you can use a depot-relative pathname to refer to any element in the same workspace. If your current

directory is not in the workspace, or you apply a command to a particular stream, using the `-s` option, you must specify elements using their depot-relative pathnames.

Selecting Files Using Filename Patterns and AccuRev Status

In many commands, you can specify the set of files to be processed using either or both of these mechanisms:

- Filename patterns (for example, `*.py`). On Windows client machines, the **accurev** program performs pattern-matching itself. On UNIX/Linux windows machines, pattern-matching is performed by the operating system.
- Filtering on AccuRev file status (for example `-m`, for “modified files”).

See [Techniques for Selecting Elements](#) on page 10.

Working with Multiple Repositories

It’s possible to have multiple AccuRev repositories active in your organization, each managed by its own AccuRev server process. For most **accurev** CLI commands, you can specify the AccuRev server to target on the command line, using the `-H` option:

```
accurev show -H pluto:6678 users
```

Note that the `-H` option follows the command name — in this example, **show** — not the program name, **accurev**.

This mechanism bypasses the **acclient.cnf** file, though the file must still exist. It does *not* override a specification in the **wspaces** file, which is described below.

Setting Up Workspaces on Multiple Repositories

If the workspaces you’ve created on your client machine are not all associated with the same AccuRev server, then you might want to use a **wspaces** configuration file. This text file records the workspace-to-server association for each of your workspaces. Each user can have a **wspaces** file. It must be located under the user’s home directory, within the **.accurev** subdirectory.

The **accurev** program uses this file to decide which AccuRev server to use:

- If the current working directory is within one of the workspace-root-directory locations listed in the **wspaces** file, the CLI command is directed to the associated server listed on the same line of the file.
- Otherwise (including the case in which no **wspaces** file exists), the CLI command is directed to first server listed in the machine’s **acclient.cnf** file. This is the default behavior that we’ve mentioned in the sections above.

For example, here is a sample **wspaces** file recording workspaces associated with servers **venus** and **pluto**:

```
gizmo_dvt_john      /usr/john/gizmo_dvt      venus  5050
frammmis_2.3maint_john /usr/john/frammmis_2.3_maint pluto  6678
```

The format of the file is:

- workspace name, which must not contain SPACE characters
- full pathname of workspace’s root directory, which must match the pathname listed by the **accurev show wspaces** command, and must not contain SPACE characters. The pathname always uses forward slashes (`/`), not backslashes (`\`), even on Windows machines.
- server machine name (or IP address)

- IP port number on which AccuRev server program listens

The last two fields must exactly match the server and port of an existing entry in the machine's **acclient.cnf** file.

Note: If your site has a single AccuRev server (or if your workspaces are all associated with a single server), there is no need for a **wspaces** file: the default behavior directs each CLI command to the server listed in your machine's **acclient.cnf** file.

File Locking in Workspaces

Exclusive file locking (serial-development mode) can be implemented at the depot level, the workspace level, or the element level (*keep -E*, *add -E*). If exclusive file locking applies to an element:

- It is maintained in the read-only state in your workspace tree when you are not actively working on it.
- If the element is active in a sibling workspace (having been processed with *anchor*, *co*, *revert*, *keep*, *move*, *defunct*, or *undefunct*), you cannot perform any of those commands on it in your workspace.

Note: In this context, workspaces are considered siblings if they promote to the same stream. If the stream hierarchy includes pass-through streams, workspaces can be siblings even if they have different parents.

- If the element is not active in any sibling workspace:
 - You can make a file element active with the *co* or *anchor* command. This makes the file in your workspace tree writable.
UNIX/Linux: only the “user” write permission is enabled, not the “group” or “other” write permission. Your *umask* setting is ignored.
 - You can use the *keep*, *move*, *defunct*, *undefunct*, or *revert* commands, which make an element active in the workspace in addition to their normal functions, without first using *co* or *anchor*.

An exclusive file lock on a file element is released when active development on the file ends in that workspace — that is, when you either *promote* or *purge* your changes.

In an anchor-required workspace, *all* elements are maintained in a read-only state when you are not actively working on them. Using such a workspace is similar to working with exclusive file locking, except that you are not constrained by elements' activity in sibling workspaces:

- You can make a file element active with the *co* or *anchor* command. This makes the file writable.
- You can make an element active with *move*, *defunct*, *undefunct*, or *revert* without having to first use *co* or *anchor*.
- If an element is active in a sibling workspace, you will get a warning message. You can override this warning and anchor the file in your workspace by using *anchor -O <filename>*.

See [Serial Development through Exclusive File Locking](#) on page 25 of the *AccuRev Concepts Manual*.

Entity Names

AccuRev deals with many kinds of named entities: files, directories, pathnames, depots, users, and so on. The table below details the restrictions on user-defined names for the various entities. Names are case-sensitive by default; for example, user **john** is not the same as user **John**. In case-insensitive depots, pathnames are case-insensitive.

Note: In the table below, “characters” means NON-NULL characters. Do not subtract 1 to account for the terminating NUL.

Kind	Maximum Name Length	Characters Allowed in Name	Character Restrictions
depot stream snapshot workspace reference tree	79	A-Z,a-z,0-9 - _ . + @ SPACE	can begin with a digit but cannot be only digits cannot contain forward-slash or backslash characters cannot begin with a dot (.)
user group	99	same as above	must begin with a letter
password	19	any	none (caution: this means that special characters, such as BACKSPACE are valid password characters)
pathname of slice (of a depot or replica)	255	name validation performed by the operating system	applied by the operating system
pathname of workspace pathname of reference tree	127	name validation performed by the operating system	applied by the operating system
segment (between slashes) of element pathname	255	name validation performed by the operating system	applied by the operating system

AccuRev User Preferences

Some *accurev* commands use AccuRev user preferences to control or modify their operation. User preferences can be supplied in either of these ways:

- As a setting in your AccuRev user preferences file: **preferences.xml** in the **.accurev** subdirectory of your AccuRev home directory. You can use the AccuRev commands *setpref* and *getpref* to maintain the contents of this file.
- As an environment variable.

If an environment variable has the same name as an entry in the user preferences file, **accurev** commands follow the environment variable and ignore the preference-file entry.

An alphabetical listing of these settings is provided under [ENV_VARS](#) on page 272 of the [AccuRev® Command-Line Reference](#) chapter. Note that you can display a listing of all AccuRev environment variables when using the command line by typing:

```
accurev help ENV_VARS
```

Element Status

Each element has a status in each workspace or stream in which it appears. Keep in mind that an element can (and usually does) have different status in different workspaces/streams.

An element's status consists of one or more of the following status indicators.

Link-related indicators:

- **(elink)** — the element is an element link.
- **(slink)** — the element is a symbolic link.

(There is no corresponding status indicator for a file element or directory element.)

- **(missing-target)** — For an element link, the target element is not present in the workspace or stream. This can occur if the target element is removed from the workspace tree by an operating system command. It can also result from an *incl -b* or *excl* command. For a symbolic link, there is no object at the target pathname.
- **(modified-target)** — For an element link, the target element has been modified (either a content change or a namespace change) in the workspace or stream.
- **(defunct-target)** — For an element link, the target element has **(defunct)** status in this workspace or stream.
- **(nonexistent-target)** — For an element link, the target element does not appear at all in this stream or workspace. This occurs when the target element is *defunct*'ed then *promote*'d to the parent stream.
- **(corrupted)** — For an element link in a workspace, AccuRev and the operating system disagree on the link target. That is, the target element recorded in the AccuRev repository differs from the target in the operating system's instantiation of the link in the workspace tree. This can occur if you modify or replace a link using operating system commands instead of AccuRev commands.

A cross-linked element (see **(xlinked)** below) gets **(corrupted)** status if AccuRev does not overwrite the element during execution of the *Include from Stream* command, because the element has **(modified)** or **(kept)** status in the workspace. This should not occur during normal operation.

Presence of the element in the workspace:

- **(defunct)** — the element has been marked for removal from the workspace stream with the *defunct* command. The element has already been removed from the workspace tree (local disk storage); it gets removed from the workspace stream (in the depot) when you *promote* the element to the backing stream.
- **(external)** — the file or directory has not been placed under version control. (It's in the workspace tree, but not in the workspace stream.) See [Specifying Ignore Patterns for External Objects](#) on page 7 in *AccuRev Technical Notes*.
- **(ignored)** — the external file or directory matches a pattern specified in either the *.acignore* file, or the *--ignore* option (for the *add*, *files*, and *stat* commands). See [Specifying Ignore Patterns for External Objects](#) on page 7 in *AccuRev Technical Notes*.
- **(excluded)** — the element does not appear in the workspace because it has been excluded, using the Include/Exclude facility. For file elements, it's more likely that the exclusion was explicitly set on the directory in which the file resides, or in a higher-level directory that includes the file. See the *incl*, *excl*, and *incldo* reference pages.

- **(xlinked)** — this version of the element appears in the workspace or stream by virtue of a cross-link (*incl -b* command) — either on the element itself or on its parent directory or a higher-level directory.
- **(missing)** — the workspace “should” include a version of this element, but doesn’t. This occurs most often when you delete version-controlled files from the workspace tree using operating system commands. If an operation causes the target of an element link to be removed from a workspace, AccuRev removes the element link, also, causing it to have **(missing)** status.
- **(twin)** — the element is one of multiple elements in the workspace that exist at the same pathname. At most one of these elements can be accessed through the pathname; the other(s) can be accessed through their unique element-IDs. See [Version Control of Namespace-Related Changes](#) on page 35 in *AccuRev Technical Notes*.
- **(stranded)** — the element is active in the workspace, but cannot be accessed through the file system. This can occur in several situations:
 - There is no pathname to the element, because the element’s directory (or a higher-level directory) was removed from the workspace or stream.
 - (dynamic stream only) There are one or more defunct elements at a given pathname, along with one non-defunct element. The defunct element(s) have **(stranded)** status.
 - The element’s directory (or a higher-level directory) is cross-linked, making another version appear at the pathname of the active version.

Changes to the element in the workspace:

- **(modified)** — the file has been modified in the workspace since the most recent *update* or *keep*. See [Optimized Search for Modified Files](#) on page 247.
- **(kept)** — a new version of the element has been created with *keep*, *move*, *defunct*, or *undefunct*, and the file has not subsequently been modified, *promote*’d to the backing stream, or *purge*’d.
- **(member)** — the element is “active” in the workspace (is in the workspace stream’s default group). The commands that create a new version, listed above, also make the element active. So do the commands *merge* and *revert*, which invoke the *keep* command. And so do the commands *anchor* and *co*, which make an element active by creating a virtual version in the workspace.

Relationship to the version in the backing stream:

- **(backed)** — the version in the workspace stream is the same as the version in the backing stream. And you have not changed the element since the last time you *promote*’d or *purge*’d it, or since the most recent *update* of your workspace.
- **(stale)** — the element needs to be updated, because the version in the backing stream has changed since the workspace’s latest *update*. And since you have not changed the element in your workspace, it can be safely updated.
- **(overlap)** — the element has changed both in the backing stream and in your workspace. This indicates that a merge is required before you can promote your changes to the backing stream. Prior to AccuRev 4.6, “underlap” files were considered to have “overlap” status.
- **(underlap)** — similar to *overlap*: the element has changed both in the backing stream and in your workspace, but the changes in your workspace have already been promoted to the backing stream. (More precisely, your version is an ancestor of the backing stream’s version.) In many cases, no *promote* is required; you can just *purge* the changes from your workspace, restoring the version that was in the backing stream at the time of the workspace’s most recent *update*. Prior to AccuRev 4.6, “underlap” files were considered to have “overlap” status.

In a new workspace, the status of every element is backed. This means that the version in your workspace is identical to the version in the backing stream. The following sections describe the other statuses.

Backed vs. Modified vs. Kept

As soon as you make a change to an element, its status changes to modified. The element's status stays “modified” until you do a *keep* or *purge* on it. If the element is purged, its status reverts to “backed”.

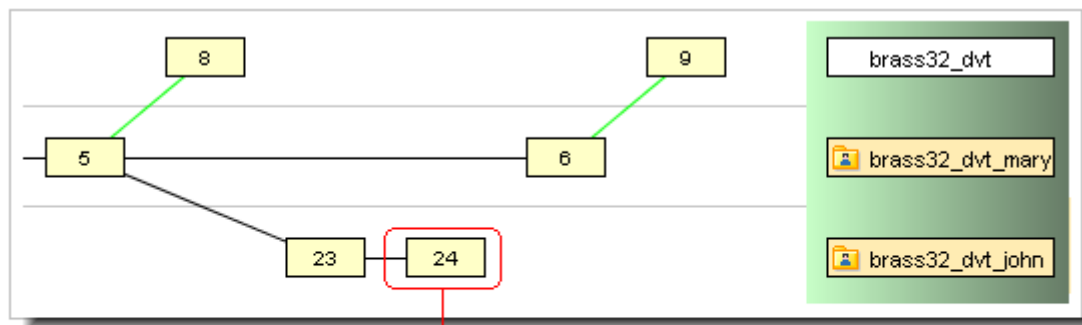
When you *add* a new element or *keep* an existing element, its status becomes kept. The element stays “kept” until you do a *purge* or *promote* on it (which changes the status to “backed”), or make a change to it (which changes its status to “modified”).

Overlap vs. Underlap

Note: the examples in this section were taken from the Version Browser in the AccuRev GUI to better illustrate the circumstances that lead to (*overlap*) and (*underlap*) statuses.

Prior to AccuRev 4.6, (*overlap*) status was defined as follows:

Version W, in a workspace or stream, has (*overlap*) status if version P in the parent stream is not an ancestor of version W.



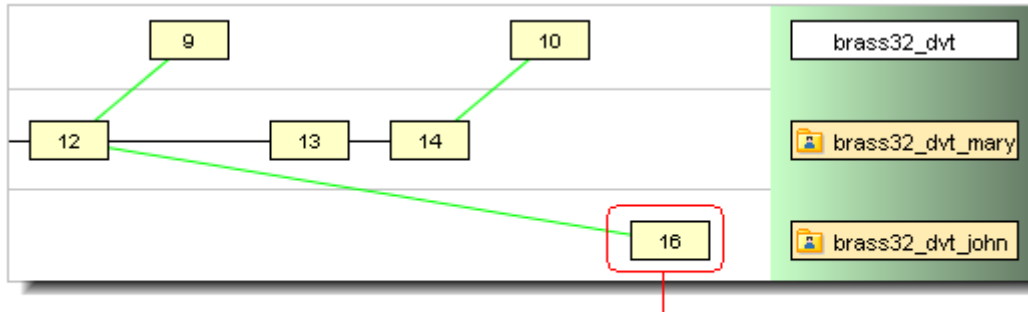
Element has (overlap) status in workspace brass32_dvt_john, because version in parent stream (brass32_dvt/9) is not an ancestor of this version

Starting in Version 4.6, AccuRev distinguishes this special case of (*overlap*):

Version P is not an ancestor of version W, but Version W *is* an ancestor of version P.

That is, the changes in the workspace or stream’s version already appear in the parent stream. This is not considered to be an (*overlap*), but a new status, called (*underlap*).

The example below illustrates a common (*underlap*) scenario: **john** brings an old version, **brass32_dvt_mary/12**, of the element into his workspace, using the *co* (GUI: *Send to Workspace*) command; this old version is an ancestor of the version currently in the backing stream of **john**’s workspace.



Element has (underlap) status in workspace brass32_dvt_john, because this version is an ancestor of the parent stream's version (brass32_dvt/10)

Techniques for Selecting Elements

The following sections describe techniques for specifying the elements to be processed by a command, based on AccuRev status, on filename, or both.

Selecting Elements Based on their Status

A few command-line options are recognized by many of the **accurev** commands, and work identically (or nearly identically) in each command. These options act as filters, selecting a subset of the depot's elements based on their element status. This makes it easy to select sets of elements for a command to operate on. You can use these filter options alone or in combination with filename arguments. Examples:

Invoke the **keep** command on all items in the current directory:

```
accurev keep *
```

Invoke the **keep** command on all items in the current directory with **(modified)** status.

```
accurev keep -m *
```

Invoke the **add** command on all items in the current directory with **(external)** status and a **.png** filename suffix.

```
accurev add -x *.png
```

Invoke the **stat** command on all items in the workspace, in all directories.

```
accurev stat -a
```

The following sections describe useful scenarios of selecting a set of elements in your workspace, based on the elements' current status. Be sure to read the section [Performance Considerations for Element-Selection Commands](#) on page 12, also.

Selecting Modified Elements

The **-m** option selects modified elements. By default, it selects all modified elements. When combined with other selection criteria, it narrows the selection to just the modified elements. This is how we obtained the earlier example with **keep**. To keep all modified elements:

```
accurev keep -m
```

To keep all modified elements in the current directory:

```
accurev keep -m *
```

To simply find out which elements match a given set of criteria, use the *stat* command. As in the previous example, to list all modified elements in the current directory:

```
accurev stat -m *
```

Selecting Kept Elements

To select all kept elements, use the *-k* option. This command promotes all the elements in the depot that you've kept:

```
accurev promote -k
```

When combined with other selection criteria, *-k* narrows the selection to just the kept elements. To promote all kept elements in the current directory:

```
accurev promote -k *
```

Selecting Members of the Default Group

Each workspace and stream has a default group, a list of its “active” elements:

- In a workspace, these are the elements that you've “officially” activated with an **accurev** command (*anchor*, *co*, *keep*, *move*, *defunct*, *undefunct*, *revert*), but have not yet promoted to the backing stream. The default group does not include elements that you've simply modified with a text editor.
- In a (non-workspace) stream, the active elements are those that have been promoted to the stream from a workspace below it (or from another non-workspace stream), but have not yet been promoted to the parent stream.

The *purge* and *promote* commands remove elements from the default group.

You can restrict a command to using the elements in the default group, by using the *-d* option. This is very useful when you're working on a large depot: determining the contents of the default group is a relatively quick operation, whereas examining all the elements in the depot is comparatively time-consuming. For instance, if you are working on a depot with 1000 files but you only have 10 files in your default group, using the options *-m -d* to select the modified elements in the default group only has to check 10 files. This will run much faster than using just the *-m* option, for which **accurev** must examine all 1000 files.

Selecting Non-Member Modified Elements

If you want to make the best use of the default group, you will periodically need to add modified elements to the default group if they aren't already in it. You can select non-member modified elements with the *-n* option. To add all non-member modified elements to the default group:

```
accurev co -n
```

Selecting Pending Elements

The end goal of any change is to promote that change to the backing stream. All changes, whether kept or not, are considered to be pending. Pending is short for “pending promotion”. The pending option is necessary because *-k* does not select modified elements and *-m* does not select kept elements. To list all elements that are pending promotion in the current workspace:

```
accurev stat -p
```

Selecting Overlapping/Underlapping Elements

Elements that you have changed and have also been changed in the backing stream since you last updated your workspace are called overlapping. Before you can promote these elements, you must merge your changes with the new changes in the backing stream. To get a list of all overlapping elements:

```
accurev stat -o
```

When you are ready to do the merges, you can do a merge of all overlapping elements very simply:

```
accurev merge -o
```

Underlap status is similar to overlap status, in that an element has changed in your workspace and also in the backing stream. With underlap, the changes in your workspace version have already been promoted to the backing stream (from another workspace, or from a stream elsewhere in the depot's stream hierarchy). In many cases, the most appropriate action is to use the *purge* command to “undo” the changes in your workspace. In other cases, a merge-promote sequence is most appropriate.

To get a list of all underlapping elements:

```
accurev stat -U
```

To purge the changes in underlapping elements, specify them on the command line or in a list-file:

```
accurev purge blink.java BlinkProj.doc
```

Note: *purge* restores the version of an element that was in the backing stream at the time of your workspace's most recent update. You might need to perform another *update* to bring the version causing the underlap status into the workspace.

Selecting Objects That AccuRev Doesn't Know About

Files and directories that haven't been processed with the *add* command are called external. These can be selected with the *-x* option. This is useful for creating new elements — that is, placing files under version control:

```
accurev add -x
```

Before doing so, it is a good idea to find out which files AccuRev considers to be external:

```
accurev stat -x
```

Selecting All Elements

To select all the elements in a depot, use the *-a* option. (If you use *-a*, you cannot also list elements on the command line.) This command displays the status of all of the elements in a depot:

```
accurev stat -a
```

The *-a* option does not include external objects.

Performance Considerations for Element-Selection Commands

The commands in the preceding sections all follow the same pattern: find all the elements in your workspace that satisfy a certain condition, and perform an operation on the selected set of elements. Many of these commands require that AccuRev consider every file in your workspace, even the ones that you haven't placed under version control (for example, editor backup files, files produced by software builds). If your workspace contains many thousands of files, such operations can be time-consuming.

These are the command options that require a full-workspace search:

- p* elements that are pending promotion to the backing stream
- m* elements that you've modified (and possibly kept, too)
- n* element that you've modified but have not kept
- o* elements with overlap status
- U* elements with underlap status
- B* status up the backing chain, including deep overlaps
- x* files that you have not placed under version control (external)
- M* elements under version control, but no file appears in your workspace (missing)

AccuRev provides optimizations that help to speed the performance of workspace and stream searches. See the *files* and *stat* reference pages for more information.

Selecting Elements Using Filename Patterns

Whenever you need to specify one or more filenames in an **accurev** command, you can:

- (*UNIX/Linux*) specify them using standard wildcards, which the shell will expand properly.
- (*Windows*) specify them using one or more filename patterns (“wildcards”). You can also combine patterns and complete filenames in the same command.

The **accurev** program recognizes the following filename patterns:

*

Matches any 0 or more characters.

?

Matches any 1 character.

[aekz]

Matches **a**, **e**, **k**, or **z**.

[e-k]

Matches **e**, **f**, **g**, **h**, **i**, **j**, or **k**. Note: do not mix uppercase and lowercase.

{one,two,seven}

Matches any of the three strings: **one**, **two**, or **seven**.

/ or \

Matches any directory-separator character, even if it's the “wrong one” for the client machine. For example, **src/do_*.java** matches **src\do_something.java** on a Windows client.

Examples:

- The following command displays the status of the files **rln_4.1.1.htm**, **rln_4.1.2.htm**, and **rln_4.1.3.htm**:

```
accurev stat rln_4.1.?.htm
```

- The following command displays the status of the three files listed above, along with others such as **rln_4.htm**, **rln_4.5.txt**, and **rln_4.1.3k-ubuntu.htm**:

```
accurev stat rln_4*.{txt,htm}
```

But it would not display the status of file **rln_4-windows.chm**.

- The following command creates new versions of five files, **color_f**, **color_g**, ..., **color_j**.

```
accurev keep color_[f-j]
```

Combining File-Status Filters with Filename Patterns

If you use a file-status filter in the same command as a filename pattern, the pattern is applied before the filter. For example:

```
-k *.doc
```

... specifies this two-step selection process:

1. Select all elements in the current directory that have the suffix **.doc**.
2. Select the subset of those elements that have **(kept)** status.

Selecting Elements Using Element-IDs

With numerous commands, you can specify a single element using its integer element-ID, which is unique within its depot. For example:

```
accurev cat -v isox/4 -e 4823
```

To determine the element-ID of an element, run the **stat -fe** or **hist** command on the element.

Using a Specific Version of an Element

Most commands have defaults for which versions to use; many of them let you specify a different version with the **-v** option. For instance, the **diff** command defaults to comparing the file in your workspace with the most recently kept version. Using the **-v** option, you can compare the file in your workspace with any version:

```
accurev diff -v 1/4 foo.doc
```

The version specification following **-v** can be either of the following:

```
<stream-name>  
<stream-name> / <version-number>
```

If your current directory is within a workspace, which establishes a particular depot as the “current depot”, then you can also use either of the following with **-v**:

```
<stream-number>  
<stream-number> / <version-number>
```

(Stream names are unique throughout the repository; but all depots use the same stream numbers. Thus, you need a current depot context to make a stream number, such as **2**, unambiguous. To do so, either you must be in a workspace, or you must specify a depot using the **-p** option.)

You can use a forward slash (/) or a backslash (\) on any platform. With UNIX/Linux shells, you’ll need to quote or escape the backslash character.

A specification that includes a **<version-number>** is completely deterministic and invariant. Once someone creates version **13** in stream **tulip_dvt**, the version specification **tulip_dvt/13** always refers to that version, no matter what happens in the future (well, almost — see the Note below).

A specification that includes only a **<stream-name>** or **<stream-number>**, not a **<version-number>**, says “the version that is currently in use by the specified stream”:

- If the element is active in that stream — that is, the element is in the stream’s default group — this means the most recent (highest-numbered) version in the stream.
- If the element is not active in that stream, the stream “inherits” its current version of the element from its parent stream. The parent stream might, in turn, inherit its current version from the next higher stream in the hierarchy.

Note: AccuRev’s TimeSafe property means that once a version is created, it can never be destroyed. If a stream is renamed (*chstream* command), a version specification can still use the old stream name — or the stream number, which never changes. If a stream is deactivated (*remove* command), its versions become inaccessible. This is not irrevocable, though: you can reactivate a stream, making its versions accessible again, with the *reactivate* command.

Getting Work Done with the CLI

AccuRev does not force you to proceed with your software development work in any particular way. The workflow described in the following sections shows how you can put the AccuRev CLI to good use, but it certainly isn’t the only way to get your work done.

Our example workflow includes these topics:

- [Creating a Workspace](#)
- [Placing Files Under Version Control](#)
- [Editing Files in a Workspace](#)
- [Checkpointing — Saving Private Versions](#)
- [Comparing Versions of a Text File](#)
- [Making Your Changes Public](#)
- [Concurrent Development — Working Well with Others](#)
- [Determining the Status of Files](#)
- [Getting Element History](#)
- [Tracking Other Users’ Work](#)
- [Incorporating Other Users’ Work into Your Workspace](#)
- [Concurrent Development — When Streams Collide](#)

Creating a Workspace

Use the *accurev mkws* command to create a new workspace. Using command-line options, you *must* specify three parameters:

- **Name of backing stream (–b option):** Each workspace is associated with (or “based on”, or “backed by”) one of the depot’s streams. If a depot represents a particular development project, then a stream represents a sub-project. The backing stream acts as a “data switchboard”, organizing the sharing of your changes to files with changes made by other members of your development team.

You can name any dynamic stream or snapshot (static stream) as the backing stream for the new workspace. You cannot choose a workspace stream — a workspace cannot be based on another workspace. For example, to specify the stream named **brass_dvt** as the backing stream:

```
-b brass_dvt
```

- **Name of new workspace (-w option):** AccuRev identifies each workspace by a simple name, which must be unique across the entire repository. That is, two workspaces cannot have the same name, even if they belong to different depots. You can specify any name; *accurev* will automatically add the suffix *_username* to it (unless you type the suffix yourself). If your username is **derek**, then these two specifications are equivalent:

```
-w brass_dvt_derek  
-w brass_dvt
```

As the above example shows, the AccuRev convention is to name a workspace after its backing stream, with the username suffix providing uniqueness. This convention makes it easy to set up a backing stream and a set of like-named workspaces, one for each user:

backing stream:

```
brass_dvt
```

workspaces:

```
brass_dvt_derek  
brass_dvt_mary  
brass_dvt_john
```

- **Location of workspace tree in file system (-l option):** A workspace’s name is a simple identifier, stored in the AccuRev repository. You must also decide where the workspace will be located on your hard disk (or elsewhere in file storage to which you have access). If you specify a pathname that does not yet exist, *accurev* creates an empty directory at that location. If you specify an existing directory, *accurev* “converts” it to a workspace — the files in the existing directory tree become versions of the elements in the backing stream (if their pathnames match), or become external files (if their pathnames don’t match those of existing elements).

Examples:

```
C:\dvt_work(Windows)  
C:\ac_workspaces\brass_dvt  
H:\Projects\Colorwheel\mary  
  
/usr/projects/mary/brass_dvt(UNIX/Linux)  
/workspaces/ColorWheel/mary
```

In addition to these mandatory specifications, there are options for controlling other aspects of the new workspace:

- **Workspace kind (-k option):** A standard/default workspace (*-kd*) contains a copy of each element in the backing stream. By default, each user can edit the files in his workspace without having to issue a “check out” command, and irrespective of the status of the file in other users’ workspaces. The *-ke* option provides exclusive file locking: after you use *co* or *anchor* to start working on a file, users in other workspaces sharing the same backing stream are prevented from working on the same file.
- **Text-file line terminator:** By default, AccuRev uses the line terminator appropriate for the client machine’s operating system whenever it copies a version of a text file into the workspace. You can force the workspace to get text files with UNIX/Linux (*-eu*) or Windows (*-ew*) line terminators.

Placing Files Under Version Control

The files in a workspace are not automatically version-controlled. After all, there are many files that you don't *want* to version-control: text-editor backup files, intermediate files and log files produced during software builds, files you may have downloaded from the Internet, etc. A file that's in a workspace but is not under version control is said to have external status.

To place one or more external files under version control, use the ***add*** command:

```
accurev add Red.java White.java Blue.java
```

(The files must already exist; ***add*** won't create an empty file for you.)

You can have ***add*** search for *all* external files — throughout the entire workspace — and convert them to elements:

```
accurev add -x
```

This also places the directories containing those files under version control (if they aren't already). Using ***add -x*** makes it very easy to convert an existing directory tree into a workspace (see [Creating a Workspace](#) on page 15), then place all the files in that directory tree under version control.

Editing Files in a Workspace

By default, the files in a workspace are always writable. You can edit the files at any time, using a text editor, an IDE, or any other application.

If you're in a workspace in which you must “check out” a file before editing it (see [File Locking in Workspaces](#) on page 5), version-controlled files are maintained in a read-only state until you invoke the ***co*** or ***anchor*** command on them.

Checkpointing — Saving Private Versions

At any time, you can keep the changes you've made in one or more files. The ***keep*** command creates an official new version of a file. This includes making a permanent copy in the depot of the file's current contents. At any point in the future, you can revert to this version. This “save it just in case” procedure is commonly called checkpointing.

But ***keep*** does not make the new version public — it remains private to your workspace. Nobody else will see your changes yet. You can keep as many private versions (i.e. checkpoint the file as many times) as you want, without affecting or disrupting other people's work.

Here's an example of the simplest form of the ***keep*** command:

```
accurev keep testrun.pl
```

You'll be prompted to enter a comment string, which can span multiple lines. It might be easier to include short comments in the command itself:

```
accurev keep -c "shorten timeout" testrun.pl
```

You can specify multiple files in a single command. If you want to keep a large number of files at once, it's probably easiest to place their pathnames in a text file (say, ***my_filelist***), and use the ***-l*** option:

```
accurev keep -l my_filelist
```

The ***keep*** command can use file-status filters. For example, this command keeps all files — throughout the entire workspace — that you've modified but not yet kept:

```
accurev keep -m
```

Comparing Versions of a Text File

The standard development procedure for a version-controlled file consists of these steps:

- Edit the file, then use the *keep* command to create a new, private version.
- Repeat the preceding step as many times as desired (or not at all).
- Make the (most recent) private version public, using the *promote* command.

As you're working on a file using this procedure, you'll often want to compare the current contents of the file with other versions. The *diff* command does the job.

For example, you might want to know, "what changes have I made to file **Red.java** since the last time I performed a *keep* on it?" This command shows the answer:

```
accurev diff Red.java
```

If you've created several intermediate versions with *keep*, you might want to ask, "what changes have I made to file **Red.java** since I started working on it?". Often, this question is answered by comparing your file with the version in the backing stream:

```
accurev diff -b Red.java
```

These are just some simple examples; you can use *diff* to compare *any* two versions of a text file element. By default, *diff* output looks like this:

```
diffing element \.\doc\procfiles.py
56a57
>     ### -f option (formerly -l):
59c60
<     if optdict['-l']:
---
>     if optdict['-f']:
62c63
<         for line in f.readlines():
---
>         for line in f():
```

Using an environment variable, you can configure *diff* to use a third-party file-difference program.

Making Your Changes Public

To make your work on a file available to others, you promote the (most recent) private version in your workspace to the backing stream. That is: *keep* creates a private version, and *promote* turns it into a public version.

(Other users, whose workspaces have the same backing stream, can incorporate your promoted versions into their workspaces with the *update* command. See [Incorporating Other Users' Work into Your Workspace](#) on page 21.)

As with *keep*, you can specify particular files to promote on the command line or place their names in a text file:

```
accurev promote testrun.pl(on the command line)
accurev promote -l my_filelist(in a text file)
```

You can specify a comment string with *-c*, but it's not required. (With *keep*, it is.)

File-status filters work with *promote*, too. For example, this command promotes all files — throughout the entire workspace — for which you’ve used *keep* to create new versions:

```
accurev promote -k
```

Concurrent Development — Working Well with Others

The essence of concurrent development (or parallel development) is enabling an entire team of developers to work with the same set of files at the same time. To avoid chaos, each user gets his own private set of files (that’s your workspace), which are copies of a “master” set of files (that’s the backing stream).

AccuRev tries to stay out of your way as much as possible. But as you proceed with development in your private workspace, you’ll often want to perform several configuration management operations:

- Determining the current status, from AccuRev’s viewpoint, of one or more elements in your workspace.
- Determine the *past* statuses — that is, the development history — of one or more elements.
- Determining what other team members are working on — and whether anyone else is working on a particular file at the same time as you.
- Incorporating other teams members’ changes into your own workspace.

The following sections describe the AccuRev commands that implement these operations.

Determining the Status of Files

Each file in a workspace has an AccuRev status, one or more keywords that describe its development state in that particular workspace. In large part, status is determined by comparing the file with the version in the backing stream. Some examples:

- If the file in your workspace is the same as the version in the backing stream, the status is **(backed)**. This occurs when you haven’t edited the file at all. It also occurs when you keep a new version of the file, then promote that version to the backing stream.
- If you edit a file without keeping it, its status is **(modified)**. If you then keep it, the status changes to **(kept)**. If you edit it again, its status becomes **(modified)** again.
- If you don’t edit a file, but another user promotes a new version to the backing stream, your version becomes **(stale)**. If you do edit such a file, its status becomes **(overlap)**, indicating that two or more users have modified the same version concurrently.

The two main commands for determining the status of files are *stat* and *files*. Use the *files* command when you’re interested in a particular set of files:

```
accurev files Red.java White.java      (status of two files in the current directory)
accurev files \.test                    (status of all files in the “test” directory)
```

Use the *stat* command when you’re interested in a particular status. For example, this command lists all the files — throughout the entire workspace — that have **(kept)** status.

```
accurev stat -k
```

The output of the *files* and *stat* commands includes one or more status flags for each file:

```
\\.src\Red.java    cwheel_dvt_derek\12    (6\12)    (modified)    (member)
\\.src\White.java  cwheel_devel\6         (6\3)     (backed)
\\.src\Blue.java   cwheel\2 10\7)         modified)
```

(The information preceding the status flags are version-IDs.)

Getting Element History

AccuRev keeps track of the complete history of each version-controlled file (or element). Changes to the AccuRev repository are structured as a set of atomic transactions. The most common transactions for an element record *keep* and *promote* actions. Transactions are also logged in other situations: when an element is first added to the depot (recorded as *create*, even though the command-name is *add*), when you rename it or move it to a different directory (recorded as *move*), when you incorporate someone else's changes into your work (recorded as *merge*), etc.

The *hist* command, in its simplest form, lists the complete transaction history of an element:

```
accurev hist White.java
```

There are many options, including listing a particular transaction:

```
accurev hist -t 43020 White.java
```

... or listing, say, just the dozen most recent transactions:

```
accurev hist -t now.12 White.java
```

You can also restrict the listing to transactions of a particular kind (say, *keep* transactions), transactions performed by a particular user, or transactions involving a particular stream.

Here are a couple of typical transactions, as listed by *hist*:

```
transaction 43020; move; 2009/10/28 14:34:44 ; user: derek
  dst : \.src\White.java
  src : \.src\W.java
  \.src\W.java 6/6 (6/6)
  ancestor: (6/5)
```

```
transaction 43019; keep; 2009/10/28 14:27:40 ; user: derek
  # improve error message
  \.src\W.java 6/5 (6/5)
  ancestor: (6/4)
```

Tracking Other Users' Work

With AccuRev, the standard way to manage a group of users working on the same project is to have a dynamic stream for the project (termed the backing stream), along with a private workspace for each user. All the workspaces are based on (or “backed by”) the common backing stream. The *wip* command (“work in progress”) shows which files are under active development across the entire project, with a workspace-by-workspace breakdown.

```
> accurev wip -s brass_dvt
\\.tools\perl\findtags.pl
  brass_dvt_mary
```

```
\\.tools\perl\reporter.pl
  brass_dvt_mary
\\.doc\chap01.doc
  brass_dvt_john
\\.doc\chap04.doc
  brass_dvt_john
\\.doc\procfiles.py
  brass_dvt_john
\\.tools\perl\reporter.pl
  brass_dvt_john
```

Incorporating Other Users' Work into Your Workspace

The set of users working on a particular project (or subproject) all have workspaces that use the same backing stream. Users make changes in their private workspaces, and preserve those private changes with *keep* commands. Then they make the changes public — that is, available to be incorporated into other users' workspaces — with *promote* commands.

Other users' work never appears in your workspace automatically. This could be destabilizing to your own work (no matter how good their code is!). Instead, you issue an *update* command when you decide to incorporate your colleagues' recently promoted changes into your work. This brings into your workspace all “new” versions of elements — versions created since the workspace's last update.

Note that the *entire* workspace is updated; you can't restrict *update* to process a particular file or directory. This reflects AccuRev's commitment to the best practice of having a workspace contain, as much as possible, a “matched set” of versions — not “old” versions of some elements and “new” versions of others.

Accordingly, the *update* command doesn't accept any filename or directory-name arguments:

```
accurev update
```

You can update your workspace as often or as infrequently as you wish. You never have to worry about “clobbering” files that you're currently working on — *update* skips over files that you have made “active” in your workspace with *keep* (or several other *keep* commands). It's even careful not to overwrite files with changes that you haven't yet preserved with *keep*.

If you need to update files that you're currently working you, you may be able to use *update -m* to accomplish this. See [Handling of \(modified\) Files / Merge on Update](#) on page 264.

Concurrent Development — When Streams Collide

In a concurrent development environment, it's inevitable that at some point, two or more users will work on the same file at the same time. That is, each user makes changes to a copy of the file in his private workspace. So the users are not *literally* modifying the same file, and there is no issue of users overwriting each other's private changes.

But the “clobbering” issue does arise when the users want to make their changes public, by promoting their private versions of the same file to the shared backing stream. If you've made changes to a file in your workspace, and another user promotes a new version of that file to the backing stream, your file's status becomes (**overlap**). You can continue working on the file as long as you wish, saving intermediate versions with *keep*. Before you *promote* your work to the backing stream, you must merge the backing-stream version with your version. This creates a version in your workspace that includes everyone's work — no one's changes get “clobbered”. You can then promote the merged version to the backing stream.

A typical invocation of the *merge* command is simple:

```
accurev merge Blue.java
```

This merges the file **Blue.java** in your workspace with the version currently in the backing stream. Often, you and your colleague(s) will modify different sections of the same file. In this case, the merge process is completely automatic, and you need only *keep* the merged file as a new version in your workspace:

```
Automatic merge of contents successful. No merge conflicts in contents.  
Actions: keep, edit, merge, over, diff, diffb, skip, abort, help  
action ? [keep]
```

If both versions being merged have a change to the same line, *merge* includes *both* changes, and you have to manually edit the results to resolve the conflict. For example, you and a colleague may have made conflicting changes to the same variable setting. That part of the merged file might look like this:

```
<<<<<< Your_Version  
    int retValue = -1;  
=====  
    int retValue = ERROR_NO_COLOR;  
>>>>>> Backing_Version
```

merge puts in the separator lines and the “Your_Version” and “Backing_Version” annotations. You edit out all this extra text, leaving just the correct assignment of the **retValue** variable, say:

```
    int retValue = ERROR_NO_COLOR;
```

When you’ve fixed all such conflicts, you *keep* the merged file as a new version in your workspace.

The preceding paragraphs describe AccuRev’s command-line merge algorithm. Alternatively, you can set an environment variable to have *merge* invoke a third-party merge program.

What About All the Other Commands?

The preceding sections focus on the day-to-day AccuRev tasks — and the *accurev* commands — that you are most likely to perform as a developer. The remainder of this chapter provides an overview of the entire CLI, with the discussion organized into broad functional categories. Each command will be discussed in much less depth than in the preceding sections. For full details, see the chapter [AccuRev® Command-Line Reference](#) on page 31.

Managing a Depot’s Stream Hierarchy

Creating Data Structures	
Command	Description
<i>mkdepot</i>	create a new depot
<i>mkref</i>	create a new reference tree
<i>mksnap</i>	create a new snapshot
<i>mkstream</i>	create a new stream

<i>mkws</i>	create a new user workspace
Maintaining Data Structures	
Command	Description
<i>chdepot</i>	change the properties of a depot
<i>chref</i>	change the name and/or definition of a reference tree
<i>chslice</i>	change the location of a slice
<i>chstream</i>	change the name and/or definition of a stream
<i>chuser</i>	rename or relicense a user
<i>chws</i>	change the name and/or definition of a workspace
<i>reactivate</i>	restore a reference tree, stream, user, or workspace to active service
<i>remove</i> , <i>rmws</i>	remove a reference tree, stream, user, group, or workspace from active service

The repository can contain any number of depots, each of which is a distinct version-controlled directory hierarchy. Each depot contains a stream hierarchy, which structures the development process for the files in that depot. A depot’s stream hierarchy consists of dynamic streams, snapshots, workspaces, and reference trees.

Most of the management commands in this category begin with “mk” (create/make a data structure) or “ch” (change the specifications of an existing data structure).

The ***mkdepot*** command creates a new depot. You can rename an existing depot (***chdepot***), or change the location of the depot’s on-disk storage (***chslice***).

The ***mkstream*** command creates a new dynamic, pass-through, or gated stream. You can change the specifications of an existing stream: its name, its location in the stream hierarchy, and (for dynamic streams) its basis time (***chstream***).

The ***mksnap*** command creates a new snapshot. Since a snapshot is, by definition, immutable, there is no “change snapshot” command.

The ***mkws*** command creates a new workspace. You can change the specifications of an existing workspace: its name, its location in the stream hierarchy, its location in your computer’s (or your network’s) disk storage, and several additional parameters (***chws***).

The ***mkref*** command creates a new reference tree. You can change the specifications of an existing reference tree (***chref***).

The ***remove*** and ***rmws*** command remove these data structures from use.

Managing and Creating New Versions of Files

Command	Description
<i>add</i>	add a new element to a depot
<i>anchor</i>	add an element to the default group of a workspace
<i>chmod</i>	change the access mode of an element (UNIX/Linux-specific)
<i>co</i>	(check out) add an element to the default group of a workspace
<i>defunct</i>	remove an element from a stream
<i>demote</i>	pull back a version from the parent stream to a child stream
<i>keep</i>	create a new version of an element
<i>ln</i>	create an element link or symbolic link
<i>merge</i>	merge changes from another stream into the current version of an element
<i>move, mv</i>	move or rename elements
<i>patch</i>	incorporate the changes from a given workspace into the current version
<i>pop</i>	copy files into a workspace or reference tree
<i>promote</i>	propagate a version from one stream to another stream
<i>purge</i>	undo all of a workspace's changes to an element
<i>revert</i>	"undo" a promote, demote, or purge transaction
<i>start</i>	create a command shell in a workspace or reference tree
<i>touch</i>	update the timestamp of a file
<i>undefunct</i>	restore a previously removed element to a stream
<i>update</i>	incorporate other people's changes into your workspace

The idea that an element is either active or passive in your workspace is essential to understanding many of the commands in this category. Typically, as you develop a file, you'll **keep** one or more versions, then **promote** the most recent one, then **keep** another version, then **promote** that one, etc. During the "keep phase" of this cycle, your workspace has a private version of the file, containing changes that don't exist anywhere else. The element is said to be active in your workspace. When you promote a version to the backing stream or demote a version to a child stream, the element becomes passive in your workspace. In this state, the public version of the element in the backing stream is the same as the version in your workspace.

Note: the elements that are currently active in your workspace are said to be in the workspace's default group.

The **add** command (see [Placing Files Under Version Control](#) on page 17 above) places a file under version control. That is, it converts an ordinary file in your workspace into a new AccuRev element. The new element becomes active in your workspace. The **ln** command creates a link object — one that points to an element (element link) or one that contains a pathname (symbolic link) that may or may not indicate an element.

The **keep** command (see [Checkpointing — Saving Private Versions](#) on page 17) creates a new version of a element in your workspace and makes the element active (if it isn't already). Several other commands make an element active in your workspace:

- The **defunct** command removes an element from your workspace. The **undefunct** command restores a previously **defuncted** element to your workspace.
- The **merge** command (see [Incorporating Other Users' Work into Your Workspace](#) on page 21) creates and **keeps** a new version of a file — and so does the **patch** command. The new version combines the contents of the file in your workspace and with *all* the changes in another version (**merge**) — or just with the most recent changes in another version (**patch**).
- The **move** (or **mv**) command changes the pathname of an element — renaming it within the same directory or moving it to another directory in the same workspace/depot.
- The **revert** command creates a new version by performing a “subtractive merge” of the specified version from the version in your workspace. See the description of [revert](#) on page 215 for details. To “revert” to the version from the backing stream, use the **purge** command instead (described below).

All of the above commands make a change to the element, and record that change as a new version in the workspace. (Yes, even **defunct** creates a new version, recording the removal of the element.)

The following commands also make an element active, creating a new version in the workspace. But these commands don't record any new change to the element; they merely transition the element from passive to active:

- The **anchor** command takes an element that is currently passive (a version is being inherited from the backing stream) and declares it to be active. Note that this doesn't make any change to the file in the workspace.
- The **co** (“checkout”) command extends **anchor** by enabling you to make *any* historical version of an element — not just the current version — active in your workspace. The **co** command copies that version from the repository to your workspace, enabling you to examine and/or edit it.

The **promote**, **demote**, and **purge** commands transition an element from active status to passive status in the from-stream. **promote** (see [Making Your Changes Public](#) on page 18) takes a private version — created by **keep** or **move** or **defunct**, etc. — and makes it public by sending it to the backing stream. **demote** removes (or pulls back) a version from the default group in a parent stream and makes that version active in a child stream. **purge** effectively discards all the private versions of the element you've created recently; your workspace reverts to the version it was using before you made the element active.

The **update** command (see [Incorporating Other Users' Work into Your Workspace](#) on page 21) copies recently-created versions into your workspace, replacing older files with newer files. It only updates elements that are passive in your workspace, leaving alone elements that are active unless you use **update -m**. The **pop** command is designed to “fill in the gaps”: it copies in the appropriate version of specified elements that are currently missing from the workspace.

The **chmod** command manipulates the UNIX/Linux-level executable bits on an element. The **touch** command updates the timestamp on a workspace file. This can affect the results of several file-status

commands that use timestamps to optimize their performance. The **start** command launches a new command shell, with the current directory set to a specified workspace.

Getting Status Information

Command	Description
<i>anc</i>	determine the ancestor of a version
<i>annotate</i>	indicate the origin of each line of a text file
<i>cat</i>	get the contents of a version of an element
<i>diff</i>	compare two versions of an element
<i>files</i>	show the status of elements
<i>getproperty</i>	show the properties for streams, users, or groups
<i>hist</i>	show the transaction history of elements or an entire depot
<i>info</i>	show basic information about the current session
<i>mergelist</i>	determine which versions need to be promoted between streams
<i>name</i>	list the name of the element with the specified element-ID
<i>patchlist</i>	list versions that need to be patched into the workspace's version
<i>show</i>	list objects of a particular kind (depots, for example)
<i>stat</i>	show the status of AccuRev elements or file system objects
<i>translist</i>	list transactions containing versions that need to be promoted
<i>type</i>	alias for cat command
<i>wip</i>	report work-in-progress for workspaces backed by a stream

The **info** command lists basic data about your AccuRev setup: username, client and server machine information, workspace and backing streams, etc.

The **show** command lists the names of items in the repository: depots, streams, workspaces, etc.

The **files** and **stat** commands (see [Determining the Status of Files](#) on page 19) list the status of elements in a workspace, or in a stream. The **name** command lists the pathname of an element, given its unique element-ID. This is particularly useful when an element has been renamed, and so appears under different names to different users. The **anc** command has options to determine several kinds of ancestor versions of a specified version (or the common ancestor of two versions). The **wip** command (see [Tracking Other Users' Work](#) on page 20) lists the files under active development in the entire set of workspaces based on a particular stream.

The **hist** command lists the transaction history of individual elements, or of entire streams or depots.

The *cat* (or *type*) command retrieves the contents of a specified version of a file. The *annotate* command lists the contents of a specified version, indicating information about how each line of the file was created or modified.

The *diff* command compares two versions of a text file.

The *translist* command considers the set of elements that have versions pending promotion in a particular workspace or stream; it lists the transactions that created those versions. The *mergelist* command lists the files that need to be merged from one specified stream to another. Similarly, the *patchlist* command considers two versions of a text file, and lists all the individual versions that have changes present in one version but not the other.

Include/Exclude Rule Facility

Command	Description
<i>clear</i>	remove an include/exclude rule
<i>incl</i>	set rule to include elements in a workspace or stream
<i>includo</i>	set rule to include a directory, not its contents, in a in a workspace stream
<i>excl</i>	set a rule to exclude elements from a workspace or stream
<i>lsrules</i>	show the include/exclude rules for a workspace or stream
<i>mkrules</i>	set or clear multiple include/exclude rules for a workspace or stream

AccuRev's include/exclude facility makes it easy to include just the files you need in a workspace or stream. The *incl*, *includo*, and *excl* command create these rules. The *clear* command deletes a rule. The *lsrules* command lists the rules. The *mkrules* command specifies multiple include/exclude rules to apply in a single file.

Administration

Command	Description
<i>archive</i>	prepare to transfer version container files to offline storage
<i>backup</i>	backs up the AccuRev metadata
<i>diag</i>	display performance diagnostics
<i>mktrig</i>	activate a trigger in a depot
<i>mkreplica</i>	add a depot to a replica repository
<i>reclaim</i>	remove archived version container files from gateway area
<i>replica sync</i>	synchronize a replica repository

Command	Description
<i>rmproperty</i>	remove properties from a stream, user, or group
<i>rmreplica</i>	remove a depot from a replica repository
<i>rmtrig</i>	deactivate a trigger in a depot
<i>setproperty</i>	create properties for a stream, user, or group
<i>synctime</i>	synchronize system clock on client computer to server computer
<i>unarchive</i>	restore version container files that were previously archived
<i>upgrade_client</i>	download the AccuRev client installation package

AccuRev has remarkably little administrative overhead. There's no need to stop the AccuRev Server process, even when using the **backup** command to back up the AccuRev metadata.

The *mktrig* and *rmtrig* commands maintain the triggers, which control users' ability to make changes to depots.

The *synctime* command adjusts a client machine's system clock to match that of the AccuRev server machine.

The *diag* command displays AccuRev performance figures.

The *archive*, *unarchive*, and *reclaim* commands manage the archiving of storage container files that can be moved to offline storage — for example, because they are no longer needed.

At a site that uses a replica of the master AccuRev repository, the *mkreplica* and *rmreplica* commands maintain the set of depots that are replicated at that site. The *replica sync* command performs a manual synchronization of the replica repository with the master repository.

Managing Users and Security

Command	Description
<i>addmember</i>	change group membership
<i>chgroup</i>	rename a group
<i>chpasswd</i>	change the password of a user
<i>chuser</i>	rename or relicense a user
<i>disable_ssl</i>	turn off SSL security
<i>enable_ssl</i>	accept SSL certificate
<i>get_certificate</i>	get new SSL certificate
<i>ismember</i>	check if named user is a group member

Command	Description
<i>lock</i>	lock a dynamic stream against promotions
<i>login</i>	log in to an AccuRev Server
<i>logout</i>	log out from an AccuRev Server
<i>lsacl</i>	show access control list entries
<i>mkgroup</i>	create a new group of users
<i>mkuser</i>	register a new username
<i>rmmember</i>	remove a user from a group
<i>rmproperty</i>	remove properties from a stream, user, or group
<i>setacl</i>	create or remove an access control list entry
<i>setproperty</i>	create properties for a stream, user, or group
<i>unlock</i>	unlock a dynamic stream to enable promotions

The AccuRev database's **site** schema contains a registry of users (with optional password protection) and user groups. Access to repository data structures is controlled by stream locks and ACLs (access control lists). Access to particular CLI commands is controlled by triggers.

The **mkuser** command allocates a unique numeric user-ID, and assigns it a new username; it optionally sets a password for the user. The **chuser** and **chpasswd** commands change the name and password settings for an existing user-ID.

Similarly, the **mkgroup** and **chgroup** commands maintain the AccuRev user-group names. User membership in groups is maintained with the **addmember**, **rmmember**, and **ismember** commands.

The **lock** and **unlock** commands control stream locks, which control the promotion of versions to and from particular streams.

The **setacl** and **lsacl** commands maintain access control lists, which control users' and groups' ability to make changes to particular streams, or to particular depots.

Managing Change Packages

Command	Description
<i>cpkadd</i>	add an entry to a change package
<i>cpkdescribe</i>	list the contents of a change package
<i>cpkremove</i>	remove an entry from a change package
<i>issuediff</i>	compare two streams in terms of their change packages

Command	Description
<i>issuelist</i>	list the change packages in a stream

AccuRev change packages are implemented using the AccuWork issue management facility. Each issue record has a Changes tab, which can record a group of changes made to a set of elements. The *cpkadd* and *cpkremove* command maintain these Change tab entries, which are termed change packages. The *cpkdescribe* command lists the contents of a change package. The *issuelist* command shows which change packages (issue records) have their changes included in a particular stream. The *issuediff* command compares two streams in terms of their change packages.

2. AccuRev® Command-Line Reference

This chapter provides a detailed description of the *accurev* program, the main command-line tool in the AccuRev configuration management system. You can use this tool in a command shell (UNIX/Linux) or at a DOS prompt (Windows). You can also invoke this tool as part of a shell script or batch file, or from a scripting language such as Perl. Each invocation of the *accurev* program looks like this:

```
accurev <command-name> <options-and-arguments>
```

This chapter begins with some overview sections, including a command summary. Then, the *accurev* commands are described in detail, in alphabetical order.

Command Summary

The *accurev* program implements the following commands:

Table 1: Summary of CLI Commands

Command	Description
<i>add</i>	add a new element to a depot
<i>addmember</i>	change group membership
<i>anc</i>	determine the ancestor of a version
<i>anchor</i>	add an element to the default group of a workspace
<i>annotate</i>	indicate the origin of each line of a text file
<i>archive</i>	prepare to transfer version container files to offline storage
<i>authmethod</i>	set or display the user-authentication method
<i>backup</i>	back up the AccuRev metadata
<i>cat</i>	get the contents of a version of an element
<i>chdepot</i>	change the properties of a depot
<i>chgroup</i>	rename a group
<i>chmod</i>	change the access mode of an element (UNIX/Linux-specific)
<i>chpasswd</i>	change the password of a user
<i>chref</i>	change the name and/or definition of a reference tree
<i>chslice</i>	change the location of a slice

Table 1: Summary of CLI Commands

<i>chstream</i>	change a stream
<i>chuser</i>	rename or relicense a user
<i>chws</i>	change the name and/or definition of a workspace
<i>clear</i>	remove an include/exclude rule
<i>co</i>	(check out) add an element to the default group of a workspace
<i>cpkadd</i>	add an entry to a change package
<i>cpkdepend</i>	list the dependencies of a change package
<i>cpkdescribe</i>	list the contents of a change package
<i>cpkremove</i>	remove an entry from a change package
<i>defunct</i>	remove an element from a stream
<i>demote</i>	move a version from the parent stream to a child stream
<i>diag</i>	display performance diagnostics
<i>diff</i>	compare two versions of an element
<i>disable_ssl</i>	turn off SSL client-server security
<i>eacl</i>	configure element access control list security
<i>enable_ssl</i>	accept or reject an SSL certificate from the AccuRev server
<i>excl</i>	set a rule to exclude elements from a workspace or stream
<i>files</i>	show the status of elements
<i>get_certificate</i>	get a new SSL certificate from the AccuRev server
<i>getconfig</i>	list the contents of an AccuWork configuration file
<i>getpref</i>	list user preferences
<i>getproperty</i>	read a stream, user, or group property
<i>help</i>	display help on the AccuRev CLI
<i>hist</i>	show the transaction history of elements or an entire depot
<i>hostinfo</i>	display information about the AccuRev license server
<i>incl</i>	set rule to include elements in a workspace or stream
<i>includo</i>	set rule to include a directory, not its contents, in a in a workspace stream
<i>info</i>	show basic information about the current session
<i>ismember</i>	check if named user is a group member
<i>issuediff</i>	compare two streams in terms of their change packages

Table 1: Summary of CLI Commands

<i>issuelist</i>	list the change packages in a stream
<i>keep</i>	create a new version of an element
<i>licenses</i>	display or set AccuRev licensing information
<i>ln</i>	create or change an element link or symbolic link
<i>lock</i>	lock a dynamic stream against promotions
<i>login</i>	log in to an AccuRev Server
<i>logout</i>	log out from an AccuRev Server
<i>lsacl</i>	show access control list entries
<i>lsrules</i>	show the include/exclude rules for a workspace or stream
<i>merge</i>	merge changes from another stream into the current version of an element
<i>mergelist</i>	determine which versions need to be promoted between streams
<i>mkdepot</i>	create a new depot
<i>mkgroup</i>	create a new group of users
<i>mkref</i>	create a new reference tree
<i>mkreplica</i>	add a depot to a replica repository
<i>mkrules</i>	set or clear multiple include/exclude rules for a workspace or stream
<i>mksnap</i>	create a new snapshot
<i>mkstream</i>	create a new stream
<i>mktrig</i>	activate a trigger in a depot
<i>mkuser</i>	register a new username
<i>mkws</i>	create a new user workspace
<i>move, mv</i>	move or rename elements
<i>name</i>	list the name of the element with the specified element-ID
<i>patch</i>	incorporate a set of changes from a given workspace into the current version
<i>patchlist</i>	list versions that need to be patched into the workspace's version
<i>pop</i>	copy files into a workspace or reference tree
<i>promote</i>	propagate a version from one stream to another stream
<i>purge</i>	undo all of a workspace's changes to an element
<i>putconfig</i>	update the contents of an AccuWork configuration file
<i>reactivate</i>	restore a reference tree, stream, user, or workspace to active service

Table 1: Summary of CLI Commands

<u>rebase</u>	purge elements with changes that are already in the backing stream
<u>reclaim</u>	remove archived version container files from gateway area
<u>remove</u>	deactivate a workspace, reference tree, stream, user, or group
<u>rename</u>	alias for <i>move</i> command
<u>replica</u>	perform operations on a replica repository
<u>revert</u>	“undo” a promote or purge transaction
<u>rmmember</u>	remove a user from a group
<u>rmproperty</u>	remove a stream, user, or group property
<u>rmreplica</u>	remove a depot from a replica repository
<u>rmtrig</u>	deactivate a trigger in a depot
<u>rmws</u>	deactivate a workspace
<u>secinfo</u>	show user’s authorization level
<u>setacl</u>	set or remove an access control list entry
<u>setpref</u>	set one or more user preferences
<u>setproperty</u>	specify a name/value pair property for streams, users, and groups
<u>show</u>	list objects of a particular kind, including properties for streams, users, and groups
<u>sitelist</u>	show the transaction history of non-depot specific site information
<u>start</u>	create a command shell in a workspace or reference tree
<u>stat</u>	show the AccuRev status of elements or file system objects
<u>synctime</u>	synchronize system clock on client computer to server computer
<u>touch</u>	update the timestamp of a file
<u>trace-event</u>	filter MQTT messages from the server for specific events
<u>translist</u>	list transactions containing versions that need to be promoted
<u>type</u>	alias for <i>cat</i> command
<u>unarchive</u>	restore version container files that were previously archived
<u>undefunct</u>	restore a previously removed element to a stream
<u>unlock</u>	unlock a dynamic stream to enable promotions
<u>update</u>	incorporate other people's changes into your workspace
<u>upgrade_client</u>	download the AccuRev client installation package
<u>verinfo</u>	display the version info of unarchived elements in a depot, sorted by size.

Table 1: Summary of CLI Commands

<i>wip</i>	report work-in-progress for workspaces backed by a stream
<i>xml</i>	submit a request to the AccuRev Server in the form of an XML message
<i>ENV_VARS</i>	This is not an AccuRev command, but an argument to the help command to display a list of AccuRev environment variables and preference settings.

Command Options

Most *accurev* commands accept command-line options. *accurev* uses the C-language *getopt* library to process command-line options and arguments. This means:

- If an option takes an argument, you can either include or omit a SPACE between the option and the argument. These are equivalent:

```
accurev stat -s tulip_dvt ...
accurev stat -stulip_dvt ...
```

- In many cases, you can combine multiple options into a single token. These are equivalent:

```
accurev merge -K -o ...
accurev merge -Ko ...
```

Most options are single-dash (for example, *-a*), though the CLI does use double-dash options on occasion (*--ignore* and *--outgoing*, for example). Duplicate options (*diff -v -v*, for example) and Windows DOS-style options (*/a* instead of *-a*) are not supported.

For common usage patterns involving command options, see [Selecting Elements Based on their Status](#) on page 10.

Return Values

Unless otherwise noted, an invocation of *accurev* returns 0 on success and 1 on failure.

add

add a new element to a depot

Usage

```
accurev add [ -c <comment> ] [ -d ] [ -E <value(s)> ] [ -x ] [ -R ] [ -s ]
[ -fi | -fI ] [ --ignore ] { -l <list-file> | <element-list> }
```

Description

The **add** command converts one or more existing files, directories, and/or links in a workspace to version-controlled elements. The new elements are placed in the depot associated with your workspace. (They cannot be moved later to another depot.) Version 1 of each element is created in your workspace stream. The new elements will not appear in other streams or workspaces until you **promote** them.

In the transaction recorded in the database for an **add** command, the AccuRev operation is listed as “create”, not “add”. Transactions are listed by the **hist** command.

If a **pre-create-trig** trigger is defined for the depot, it fires before the **add** command is executed. See the [mktrig](#) reference page and [AccuRev Triggers](#) on page 77 of the *AccuRev Administrator's Guide*.

Controlling the Element Type and Exclusive File Locking State

By default, the **add** command sets the element type to **binary** if the file contains any null characters (0x00). If the file does not contain any null characters, it sets the element type to **text**.

Note: Based on this algorithm, Unicode files are usually considered to be binary, not text.

You can override **add**'s element type determination with the **-E** option, specifying **text**, **binary**, or a third element type, **ptext** (a variant of **text** that is never assigned automatically). See below for a description of how AccuRev handles **ptext** elements. You can also set the new element's exclusive file locking state: **serial** (exclusive file locking enabled) or **parallel** (exclusive file locking disabled).

By default, all files in a workspace are writable. You can edit any file at any time, and use **keep** to create new versions of them. But if you use the **-E serial** option, the file becomes read-only after you **promote** it to the backing stream. See [File Locking in Workspaces](#) on page 5.

You can change both the element type and exclusive file locking state in subsequent **keep -E** commands. This change affects the newly created version and future versions only; it does not change the type of any existing version.

To see an element's type, use the **hist -fv** or **stat -fk** command. To see an element's exclusive file locking state, use the **stat -fx** command (see the **hierType** attribute in the XML-format listing).

Using Triggers to Control the Element Type and Exclusive File Locking State

Setting of the element type or the exclusive file locking state (or both) can be automated with triggers. Settings made by triggers scripts override the **add -E** specifications, if any.

- To set the element type based on file extensions and/or file content, set a pre-operation trigger with **mktrig pre-create-trig**. AccuRev ships with a sample trigger script, **elem_type.pl**. You can customize it to recognize new file suffixes.
- To set the exclusive file locking state on a file-by-file basis, use the server-side pre-operation trigger, **admin_preop_trig**.

These triggers are fully described in [AccuRev Triggers](#) on page 77 of the *AccuRev Administrator's Guide*.

add

How AccuRev Handles File Elements of Different Types

AccuRev handles binary files very simply: when a new binary version is created (with *add* or *keep*), AccuRev simply copies the file from your workspace tree to the repository, creating a storage file. When you retrieve a binary file from the repository (for example, with *co -v* or *update*), AccuRev simply copies the storage file to your workspace tree.

Handling of text files is more sophisticated. By default:

- When a new version is created (*add* or *keep*), a new storage file is placed in the repository, with a single NL (or LF) character (hex character code **x0A**) at the end of each text line. This means that a version's storage file may have different line terminators than the file you submitted to the *add* or *keep* command.
- When a text file is copied into the workspace by an AccuRev command (e.g. *update*, *pop*), it gets the line terminators appropriate to the machine where the workspace is located: NL (**x0A**) for a UNIX/Linux machine, or CR-NL (**x0D x0A**) for a Windows machine.

On the individual element level, you can override the manipulation of line terminators by specifying the “preserve text line terminators” element type, with the *-E ptext* option. Files of this type are copied to and from the repository with no change, just like *binary* files.

On the workspace level, you can force the use of a particular line terminator when text files are copied to the workspace, using the *-e* option to *mkws* or *chws*. This applies to all elements of type *text*, but not to elements of type *ptext*.

How AccuRev Converts Existing Links to Link Elements

The *add* command converts existing link objects with (**external**) status to AccuRev elements as follows:

- An existing hard link is always converted to an AccuRev file element.
- An existing symbolic link (UNIX/Linux) or junction point (Windows) is converted to an AccuRev element-link element (elink) in these circumstances:
 - The target is a file element, directory element, or elink in your workspace, and you omit the *-s* command-line option. (If the target is a slink, you *must* use *-s* to convert the object to a slink. You cannot create an elink that points to a slink.)
- An existing symbolic link (UNIX/Linux) or junction point (Windows) is converted to an AccuRev symbolic-link element (slink) in these circumstances:
 - The target is an element in your workspace, and you specify the *-s* command-line option. (If the target is a slink, you *must* specify this option.)
 - The target is an (**external**) object in your workspace, and you specify the *-s* command-line option.
 - The target is a non-existent location within your workspace, and you specify the *-s* command-line option.
 - The target is a location outside your workspace.

Controlling the UNIX/Linux Access Mode

On a UNIX/Linux machine, if a file has any of its executable bits (user, group, other) set, then *add* automatically sets all three bits on the kept version. Otherwise, all three bits are cleared on the kept version.

You can set or clear the executable bits on subsequent versions, using the *accurev chmod* command.

add

Adding Directories to the Depot

add can create directory elements as well as file elements. For each file it processes, **add** automatically creates elements (if necessary) for the file's directory, its parent directory, and so on up to the top level of the depot. If you want to turn an empty directory into an element, you can specify it as a command-line argument, or you can let **add -x** find it automatically.

Options

-c <comment>

Specify a comment for the transaction. The next command-line argument should be a quoted string. Alternatively, the next argument can be in the form **@<comment-file>**, which uses the contents of text-file **<comment-file>** as the comment.

-d

Deny access to other users. If your site makes use of element-level security ("EACLs"), this option allows you to add elements to the depot and immediately make them inaccessible to other users, while assigning FULL access to the user adding the element. After adding the element, the adding user can then set the EACLs appropriately.

-E <value(s)>

Specify the element type: **text** (default) or **ptext** or **binary**; and/or specify the exclusive file locking state for this element: **serial** or **parallel**. To specify two (non-conflicting) values at once, separate them with a comma, but not a SPACE:

```
-E serial,binary
```

See [Controlling the Element Type and Exclusive File Locking State](#) above.

-l <list-file>

Process the elements listed in **<list-file>**. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any **<element-list>** specified is silently ignored.

<element-list>

One or more element names, separated by whitespace. If you also specify a **<list-file>** using the **-l** option, this **<element-list>** is silently ignored.

-R Recurse into each directory specified in **<element-list>**, and add all the external files in that directory subtree. You need not specify the **-x** option. (Use "." to specify the current working directory.)

-x Select all external files and directories in the workspace, except objects that match any of the patterns specified by the **.acignore** file. (See [Specifying Ignore Patterns for External Objects](#) on page 7 of *AccuRev Technical Notes* for more information.)

-fi Select all files and directories, even if they are excluded by the value of the **.acignore** file. (See the **-x** description, and [Specifying Ignore Patterns for External Objects](#) on page 7 of *AccuRev Technical Notes*.)

-fl Select only those files and directories specified by the value of the **.acignore** file.

add

--ignore

Specify a pattern of elements; external elements matching this pattern are ignored by the **add** command. The **--ignore** option can be used multiple times for the same command, but each instance can specify only one pattern. For example:

```
accurev add -x --ignore="*.obj" --ignore="*.txt" --ignore="!file1.txt"
```

Can also be used to specify the pathname of a text file that contains one or more patterns. For example:

```
accurev add -x --ignore="@c:\work\my_ignore.txt"
```

See [Specifying Ignore Patterns for External Objects](#) on page 7 of *AccuRev Technical Notes* for more information on specifying ignore patterns for external files.

Note that the patterns expressed using the **--ignore** option take precedence over patterns expressed using the **.acignore** file.

- s** Forces all symbolic link objects (at the OS level) whose targets are within the current workspace to be converted to symbolic-link elements (at the AccuRev level). See [How AccuRev Converts Existing Links to Link Elements](#) on page 37.

Examples

Create element **foo.c**:

```
> accurev add foo.c
```

Create binary element **fig1.jpg**:

```
> accurev add -E binary fig1.jpg
```

Create elements from all files in the workspace that are not already under version control:

```
> accurev add -x
```

Create directory elements from three new directories:

```
> mkdir sun linux windows
> accurev add sun linux windows
```

Create elements from all files in subdirectory **widgets** that are not already under version control:

```
> accurev add -x -R widgets
```

See Also

chmod, eacl, hist, keep, ln, mktrig, promote

Techniques for Selecting Elements on page 10

addmember

change group membership

Usage

```
accurev addmember [(<user-or-group-name>)] <group-name>
accurev addmember [(<user-or-group-name>)] [ <users-or-groups> ]
    [ (<users-or-groups> ) ] <group-name>
```

Description

The *addmember* command is used primarily to add one or more new members to a group. The new members can be an existing AccuRev usernames or existing groups. You can nest group membership to any number of levels. The group that the member(s) are added to is listed at the end of the command.

An alternative use of the *addmember* command is to remove one or more members from a group. The syntax for this command variant uses parentheses around each user or group to be removed. Items to be added or removed may be specified in any order on the command line.

AccuRev groups help to implement security, through the access control list (ACL), and also establish development roles.

Examples

Add AccuRev user **john_smith** to AccuRev group *eng*:

```
> accurev addmember john_smith eng
```

Create a new group, **qagrp**, and add it to the existing group *eng*:

```
> accurev mkgroup qagrp
> accurev addmember qagrp eng
```

Add several AccuRev users and groups to existing group *eng*:

```
> accurev addmember john james jenny project_qa eng
```

Add and remove several AccuRev users and groups from existing group *eng* (the command shown removes the user **john** and the group **project_qa**, and adds the user **joseph**):

```
> accurev addmember (john) (project_qa) joseph eng
```

Note: On UNIX/Linux systems, you will probably need to put backslashes (\) before the parentheses to escape them for the shell:

```
> accurev addmember \(john\) \(project_qa\) joseph eng
```

See Also

chgroup, *lsacl*, *mkgroup*, *mkuser*, *rmmember*, *setacl*, *show* (groups and members)

anc

Determine the ancestor of a version

Usage

```
accurev anc [ -v <ver-spec> ] [ -j | -J | -1 ] [ -fx ] <element>
accurev anc [ -v <numeric-ver-spec> ] [ -p <depot> ] [ -j | -J | -1 ]
[ -fx ] <element>
accurev anc -c -v <ver-spec> [ -fx ] <element>
accurev anc -c -v <numeric-ver-spec> [ -p <depot> ] [ -fx ] <element>
```

Description

The *anc* command determines one of the following:

- the direct ancestor (predecessor) version of a particular version
- the version that preceded a particular version in a specified stream
- the basis version corresponding to a particular version
- the common ancestor of two versions

In its simplest form (no command-line options), *anc* reports the direct ancestor of the version in your workspace for the specified element.

Version in the Workspace

Some forms of this command process “the version in your workspace”. This means the version currently in the workspace stream. *anc* always ignores the file, if any, in the workspace tree. Thus, the “anc” of a modified file element is *not* the version that you started editing.

Predecessor Algorithm

With no options, *anc* determines the predecessor of the version in your workspace. With just the *-v* option, it determines the predecessor of the specified version. The algorithm it uses in both these cases is:

1. If the specified version is a virtual version (in a dynamic stream), find the corresponding real version. (You can think of this as “traversing the green ancestry line” in the GUI’s Version Browser tool.)
2. Find the version from which the real version was derived. (This is “traversing the black ancestry line” in the Version Browser.)

Previous Occupant Algorithm

The *-I* option (“dash-one”, meaning “the previous one”) uses a different algorithm, which finds the “previous occupant” of a workspace or stream. (This algorithm is also used by the *diff -I* command.) The *-I* option finds the version that was in a workspace or stream just before the specified version was created there:

1. Consider the version specified with *-v* (default: the version in your workspace stream).
2. Determine the stream, S, to which this version belongs. Unlike with the predecessor algorithm, no virtual-version to real-version mapping takes place in this step. With the command *anc -v eagle_dvt/4 -I*, the stream is set to *eagle_dvt*.
3. Consider the transaction, T, that created the version in stream S.

anc

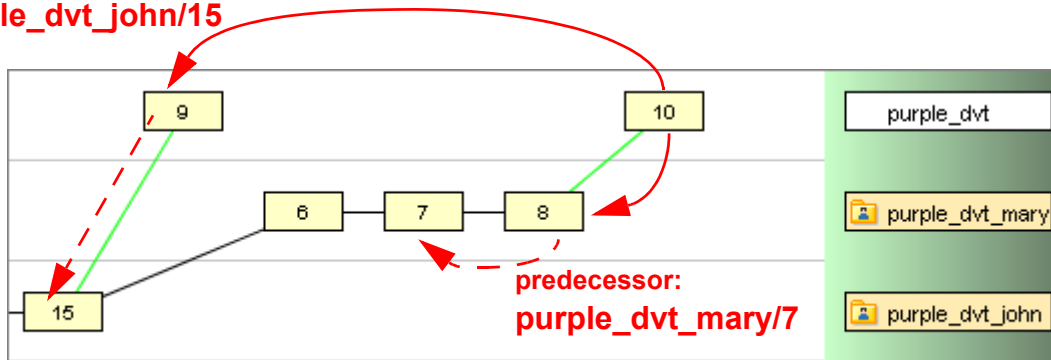
4. The result is the version that was in stream S as of transaction T-1.

For a real version in a workspace stream, this option yields the same result as the “predecessor algorithm” described above. That is, adding this option doesn’t change the command’s result. But specifying *-I* does change the result for a version created in a workspace stream by a *co* or *anchor* command; these commands create a virtual version in a workspace stream, not a real version.

For a virtual version in a dynamic stream or snapshot, the previous-occupant result may or may not be the same as the predecessor result.

previous occupant:

purple_dvt_john/15



predecessor:
purple_dvt_mary/7

Options

-c Determine the common ancestor of (1) the version in your workspace and (2) the version you specify with *-v*.

-fx Display the results in XML format.

-j Instead of determining the direct ancestor of the specified version, determine the corresponding basis version. See the [patch](#) reference page for more information on the basis version.

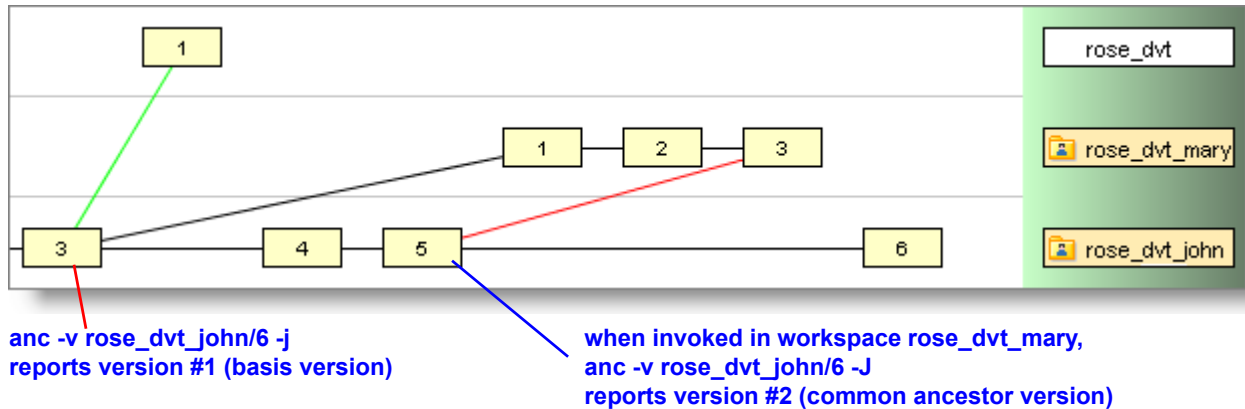
-J This option is an alternative to *-j* when used in combination with *-v*.

The command *anc -v rose_dvt_john/6 -j* finds the basis version corresponding to version **rose_dvt_john/6** of an element.

The command *anc -v rose_dvt_john/6 -J* considers two versions of the element: (#1) that same basis version and (#2) the common ancestor of version **rose_dvt_john/6** and the version in your workspace.

If version #2 is a direct descendant of version #1, then *anc* reports version #2 as the result. This occurs when a *merge* was performed to your workspace from one of the versions between the basis version and version **rose_dvt_john/6**.

anc



If version #2 is *not* a direct descendant of version #1, then **anc** reports version #1 as the result. In this case, **-J** produces the same result as **-j**.

- 1** (“dash-one”) Determine the version that was in the workspace or stream just before the specified version was created. The result is reported as a real version, even if you are “looking backward” in a dynamic stream. See [Previous Occupant Algorithm](#) above.

Note: this option is designed to be used with **-v**. Be sure to specify a complete version-ID — for example, **-v rose_dvt/14**, not **-v rose_dvt**.

- p** The depot that the specified version resides in. Used with all-numeric versions.

-v <ver-spec>, -v <numerical-ver-spec>

The version whose ancestor is to be determined. If you omit this option, **anc** uses the version in the workspace stream. If you specify the version using numbers for both stream and version (**16/1**, for example) while not in a workspace, you must also specify a depot using **-p**.

Examples

Determine the direct ancestor of the current version of element **brass.h**:

```
> accurev anc brass.h
```

Determine the direct ancestor of version **tulip_dvt/14** of element **brass.h**:

```
> accurev anc -v tulip_dvt/14 brass.h
```

Determine the basis version corresponding to the current version of element **brass.h**:

```
> accurev anc -j brass.h
```

Determine the common ancestor of the current version of element **brass.h** and version **tulip_dvt_mary\3**:

```
> accurev anc -c -v tulip_dvt_mary\3 brass.h
```

Determine the common ancestor of the current version of element **brass.h** and version **33\3** in depot **cctulip**:

```
> accurev anc -c -v 33\3 -p cctulip brass.h
```

Determine the version of **brass.h** that was in stream **eagle_dvt** just before version **eagle_dvt/4** was created by **promote**:

```
> accurev anc -v eagle_dvt/4 -1 brass.h
```

anc

See Also

[diff](#), [merge](#), [mergelist](#), [patch](#)

[Using a Specific Version of an Element](#) on page 14

*[Real Versions and Virtual Versions](#) on page 23 of the *AccuRev Concepts Manual**

anchor add an element to the default group of a workspace

Usage

```
accurev anchor [ -c <comment> ] [ -E <value(s)> ] [ -R ]
               [ -n [ -O ] ] { -l <list-file> | <element-list> }
```

Description

The *anchor* command is essentially the same as the *co* (check out) command. It adds elements to the workspace’s default group, thus preventing them from being changed by an *update* command. (The anchor keeps the file from being “swept away”— that is, overwritten — during an update.)

anchor always checks out the version in your workspace stream; thus, it never overwrites the file in your workspace with an old version of the element. By contrast, the *co* command supports the *-v* and *-t* options; this checks out an old version and overwrites the file in your workspace with a copy of that version.

A typical use of *anchor* is after an *update* command complains about unanchored files. Anchoring all such files enables a subsequent *update* to succeed.

Exclusive File Locking and Anchor-Required Workspaces

See [File Locking in Workspaces](#) on page 5.

Undoing an ‘anchor’ Command

You can undo an *anchor* command by issuing a *purge* command. This removes the element(s) from the workspace’s default group. But be careful — *purge* also discards any changes you may have made to the file(s) since *anchor*’ing them.

Options

-c <comment>

Specify a comment for the transaction. The next command-line argument should be a quoted string. Alternatively, the next argument can be in the form *@<comment-file>*, which uses the contents of text-file *<comment-file>* as the comment.

-E <value(s)>

Change the exclusive file locking state for this element: *serial* or *parallel*. See [File Locking in Workspaces](#) on page 5 of the *AccuRev CLI User’s Guide*.

-n Select only modified elements that are not already in the default group.

-O When used with the **-n** option, overrides timestamp optimization. When **-O** is used, *all* files are considered, even those whose modification times have *not* changed since the last time the workspace was updated or searched for modified files. Having to check all files, regardless of modification time, slows *anchor* performance. In anchor-required workspaces, overrides the warning message encountered when operating on an element that is active in a sibling workspace.

-R Recurse into subdirectories.

anchor

-l <list-file>

Process the elements listed in <list-file>. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any <element-list> specified is silently ignored.

<element-list>

One or more element names, separated by whitespace. If you also specify a <list-file> using the **-l** option, this <element-list> is silently ignored.

Examples

Anchor all the modified files in the workspace that are not already in the default group:

```
> accurev anchor -n
```

Anchor two particular files:

```
> accurev anchor blue.c red.c
```

See Also

co, *revert*

Techniques for Selecting Elements on page 10 of the *AccuRev CLI User's Guide*.

annotate indicate the origin of each line of a text file

Usage

```
accurev annotate [ -v <ver-spec> ] [ -f<format(s)> ] <element>
[ -E <encoding> ]
```

Description

The *annotate* command lists the entire contents of a particular version of a text file. It prefixes each line with one or more of the following: the user who created the line, the transaction in which the line was added or most-recently modified, the timestamp of that transaction, the version-ID of the version created in that transaction.

By default, *annotate* lists the current version of the file in your workspace. Use the *-v* option to specify any other version. The *-f* option specifies which annotations to include, and in which order.

Options

-v <ver-spec>

Display a particular version of the element, instead of the version in your workspace stream. See [Using a Specific Version of an Element](#) on page 14 of the *AccuRev CLI User's Guide* for a description of the forms that <ver-spec> can take.

-f <format(s)>

Use one or more of the following format letters:

t: (“transaction”) The transaction in which this line was added to the file, or was most recently modified.

u: (“user”) The user who performed that transaction.

d: (“date”) The timestamp of that transaction.

v: (“version”) The version-ID of the file version that was created in that transaction.

x: (“XML”) Display the output in XML format.

You can specify multiple format options. For example, *-fut* annotates the text lines with the user and transaction number, in that order. If you do not specify the *-f* option, AccuRev executes the command as if you had specified *-ftud*.

-E <encoding>

Indicates the encoding of the element against which you are executing the *annotate* command. Valid values include supported encodings, such as UTF-8, ISO-8859-1, EUC-JP-MS, and CP932. See the "Code Page Support" appendix in the *AccuRev Administrator's Guide* for a complete list of supported encodings.

Note that the *-E* option is ignored unless *-fx* is also specified.

Examples

For each line in file **factors.py**, display the originating transaction and the user:

```
> accurev annotate -ftu factors.py
1 john          def gcf(big, small):
1 john          """
1 john          find the greatest common factor of two numbers
1 john          """
1 john
1 john          # special cases
1 john          if big == small: return big
21 mary         # oops, wrong order
1 john          if big < small:
1 john              big, small = small, big
1 john
13 mary         # reduce, using the classic algorithm
1 john          while big % small > 0:
1 john              big, small = small, big % small
1 john
13 mary         # return greatest common factor
1 john          return small
1 john
1 john          def lcm(big, small):
1 john          """
1 john          find the least common multiple of two numbers
1 john          """
13 mary         return big * small / gcf(big,small)
1 john
1 john          def prime_factors(n):
1 john          """
1 john          return a list of the prime factors of a number
1 john          """
21 mary         factors = []
```

See Also

cat

archive prepare to transfer version container files to offline storage

Usage

```
accurev archive [ -E <element-type(s)> ] [ -i ] [ -p <depot> ]
  [ -a | -I <stream-category(s)> ] [ -s <stream> ]
  [ -t <transaction-range> ] [ -c <comment> ] [ -R ]
  { [ -Fx ] -l <list-file> | <element-list> }
```

Description

The *archive* command processes versions of one or more file elements, shifting the versions' container files from *normal* status to *archived* status. It also moves the container files from the depot's file storage area to a special gateway area (located under the depot directory).

archive determines the set of versions to archive as follows:

- Start with a particular set of file elements, which you specify as command-line arguments in the *<element-list>*, or in a list-file (plain-text or XML format). You can include directories in this list; in this case, use the *-R* option to include the recursive contents of those directories.
- Optionally, take the subset of versions whose element type matches the specification made with *-E*. (Note that different versions of an element can have different element types.)
- Optionally, take the subset of versions that were created in a particular stream (*-s*, for example, your current workspace stream). You can also archive versions from all streams in the depot (*-a*).
- Optionally, take the subset of versions created in a specific transaction, or range of transactions:
 - single transaction: *-t <number>*
 - range of transactions: *-t <number>-<number>*
- In addition to the multiple subsettings of versions described above, you can use the *-I* option to *include* versions in the set, based on where in the stream hierarchy they are referenced. For example, you can include versions that were not originally included in the set because they are referenced by one or more snapshots.

Dry Run Capability

Using the *-i* option (in addition to the other options described above) generates an XML-format listing of the final subset, but does not perform any actual archiving work. It is highly recommended that you do this before actually archiving any versions.

For a complete description of archiving, see [Archiving of Version Container Files](#) on page 37 of the *AccuRev Administrator's Guide*.

Options

- i* List the versions that would be archived (in XML format), but don't actually archive them.

archive

-c <comment>

Specify a comment for the transaction. The next command-line argument should be a quoted string. Alternatively, the next argument can be in the form `@<comment-file>`, which uses the contents of text-file `<comment-file>` as the comment.

-E <element-type(s)>

Restrict the archive operation to elements of one or more element-types: **text**, **binary**, and/or **ptext**. To specify multiple types, use a comma-separated list (default: **all** specifies all element types). Examples:

```
-E ptext,binary
-E "text, ptext"
```

-p <depot>

Select the depot to archive (default: the depot of the current workspace).

-s <stream>

Restrict the archive operation to versions in the specified stream (or snapshot, or workspace). Default: archive versions in the current workspace stream.

If you use this option, you must identify the elements to be processed using depot-relative pathnames.

-I <stream-category(s)>

Include additional versions in the archive operation, from one or more categories of the depot's streams. (Note that a version can fall into more than one of these categories.) Specify multiple categories as a comma-separated list.

- **depot** — From each stream in the depot, include all versions except the last one (most recently created). This includes dynamic streams, snapshots, and workspace streams.
- **snapshots** — Include versions that are in one or more snapshots, but are not currently active in any dynamic stream.
- **deactivated** — Include versions that are the currently selected versions in one or more dynamic streams, snapshots, or workspace streams that have been deactivated with the **remove** command.
- **all** — same as **-a**; allows you combine **-a** and **-I** options.
- **allInHierarchy** — Include all versions of a specified element underneath the specified stream hierarchy, except for those versions that are shared with another hierarchy. If no element is specified, all elements in the specified stream hierarchy are included.
- **unsafe** — Include versions that AccuRev would normally prevent you from archiving. This “force” option implies that you realize the potential complications that could arise from archiving these versions.

-a Archive *all* versions of the element(s), even versions that are currently visible in users' workspaces and active streams. This option overrides (and is mutually exclusive with) the **-I** option.

-t <transaction>[.<count>]

archive

-t *<transaction>* - *<transaction>*[*<count>*]

The first form specifies one transaction, or a specified number of transactions up to (i.e. preceding) and including a particular transaction, for example: **23695** or **now.5**.

The second form specifies a range of transactions; the optional suffix truncates the listing after the most recent *<count>* transactions. You may need to use quotes in composing the argument following **-t**; the entire argument must be interpreted by the command shell as a single token. You cannot use the **now** or **highest** keyword in the interval specification.

For the **archive** command, a transaction can be represented in the following ways:

- Time keyword: **now**
- Transaction number as a positive integer: e.g. **146** or **23965**
- Transaction number keyword: **highest**

-R Recurse into each directory specified in *<element-list>* or *<list-file>*.

-Fx Signals that *<list-file>* (see the **-l** option) is an XML-format file, not a flat text file. Example:

```
<elements>
  <element location=".\\dir07\\sub01\\file01.txt"/>
  <element location=".\\dir07\\sub01\\file02.txt"/>
  <element location=".\\dir07\\sub01\\file14.txt"/>
</elements>
```

The **-Fx** option must precede the **-l** option on the command line.

-l *<list-file>*

Process the elements listed in *<list-file>*. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any *<element-list>* specified is silently ignored.

<element-list>

One or more element names, separated by whitespace. If you also specify a *<list-file>* using the **-l** option, this *<element-list>* is silently ignored.

Examples

Note: In the XML-format listings below, the **v** and **r** attributes report the virtual version and real version, respectively.

List the versions with element-type **binary**, created in the current workspace and located below the current working directory, that are eligible for archiving:

```
> accurev archive -E binary -i -R .
<elements>
  <e
    location="/pngs/sb/sb_dftgrp_1.png"
    v="5/1"
    r="5/1"/>
  <e
```

archive

```
    location="/pngs/sb/sb_dftgrp_2.png"
    v="5/2"
    r="5/2"/>
<e
    location="/pngs/sb/sb_dftgrp_2.png"
    v="5/1"
    r="5/1"/>
<e
    location="/pngs/sb/sb_dftgrp_3.png"
    v="5/3"
    r="5/3"/>
    ...
</elements>
```

List the versions of element **sb_dragdrop_3.png**, created in the current workspace in transactions 1-25, that are eligible for archiving:

```
> accurev archive -i -t 1-25 sb_dragdrop_3.png
<elements>
  <e
    location="/pngs/sb/sb_dragdrop_3.png"
    v="5/2"
    r="5/2"/>
  <e
    location="/pngs/sb/sb_dragdrop_3.png"
    v="5/1"
    r="5/1"/>
</elements>
```

Archive the versions listed in the preceding example:

```
> accurev archive -t 1-25 sb_dragdrop_3.png
```

```
*** WARNING! *** WARNING! ***
```

```
The archive command removes files from the depot. If you do not back up
the archives created by this command, the files may be lost permanently.
```

```
...
```

```
Are you sure you want to proceed [yes/no] ? [] yes
```

```
Archiving complete.
```

```
Make sure you store the archives in a safe place for
future use. Backup is recommended.
```

```
The archives were stored to:
```

```
Archive dir : C:/Program Files/AccuRev/storage/depots/img/archive_gateway/out
```

```
Machine      : biped
```

```
Transaction  : 33
```

List the versions of element **file_8.txt** in the **s1** stream hierarchy that will be archived if you specify **-I allInHierarchy**. Note that versions that are used in another stream hierarchy are not listed and will not be

archive

archived. In this example, version v="1/1", r="5/1" is used in stream S2, and so will be excluded from the proposed archive.

```
> accurev archive -i -I allinhierarchy -s s1 file_8.txt

<e
  location="/file_8.txt"
  v="2/2"
  r="5/2"/>
</elements>
```

List the versions of element file_8.txt in the s1 stream hierarchy that will be archived if you specify **-I allInHierarchy,unsafe**. Note that versions that are used in another stream hierarchy are now listed and will be archived.

```
> accurev archive -i -I allinhierarchy,unsafe -s s1 file_8.txt

<elements>
  <e
    location="/file_8.txt"
    v="2/2"
    r="5/2"/>
  <e
    location="/file_8.txt"
    v="1/1"
    r="5/1"/>
</elements>
```

See Also

reclaim, unarchive

Techniques for Selecting Elements on page 10

Archiving of Version Container Files on page 37 of the *AccuRev Administrator's Guide*.

authmethod set or display the user-authentication method

Usage

```
accurev authmethod [ accurev_login | custom ]
```

Description

The *authmethod* command changes the user-authentication method used by the AccuRev Server, if you enter one of these keywords as a command-line argument:

accurev_login
custom

With no command-line argument, *authmethod* displays the method currently in use.

AccuRev supports the following user-authentication methods

- **accurev_login**: you are authenticated through an explicit login to the AccuRev Server.
- **custom**: you are authenticated through an explicit login, with a script processing your username/password combination.

For details on these methods, see [User Authentication](#) on page 57 of the *AccuRev Administrator's Guide*.

Examples

Display the current user-authentication method:

```
accurev authmethod
```

Set the user-authentication method to utilize the AccuRev Server's own user registry:

```
accurev authmethod accurev_login
```

See Also

info, *login*, *logout*, *secinfo*

[User Authentication](#) on page 57 of the *AccuRev Administrator's Guide*

backup

back up the AccuRev metadata

Usage

```
accurev backup <backup-file-spec>
```

Description

The **backup** command performs a backup of the AccuRev metadata stored in the database as of the time the command is issued. The resulting file is saved under the specified name in the **site_slice\backup** directory. You can override the default file location by specifying either of:

- a full path and file name when invoking the **backup** command, or
- an alternate value for BACKUP_LOC in **acserver.cnf**.

After the metadata backup has finished, you can proceed to make backup copies of all the storage files in the repository. You need not make these copies at the same time, or even on the same day. But you eventually must make copies of *all* the files in the repository. Users can continue to perform regular AccuRev operations while you're making the backup copies of the repository files.

CAUTION: Do not issue the **backup** command while you are running a program that makes a backup copy of repository files. This can place incorrect data into the backup copy.

Restoring a Backup

At any subsequent time, you can restore a backup copy of the repository. This involves:

- Restoring the files from the backup medium to their proper locations in the **site_slice** and *<depot>* directories.
- Using the administrative utility **maintain** to synchronize all the files.

This procedure returns the repository to its state at the time you ran the **backup** command.

Backup Prior to Version 5.0

In versions prior to 5.0, the **backup** command was called **backup mark**, and it declared a “checkpoint” of the AccuRev repository, which involved making copies of certain repository files in preparation for backing up AccuRev data. This command produces an error message in AccuRev 5.0.

*Important! Scripts that make use of the **backup mark** command will fail when run against AccuRev 5.0 data; you will need to change them.*

See Also

[Backing Up the Repository](#) on page 6 of the *AccuRev Administrator's Guide*.

[Restoring the Repository](#) on page 8 of the *AccuRev Administrator's Guide*.

cat get the contents of a version of an element

Usage

```
accurev cat [ -v <ver-spec> ] [ -p <depot-name> ] { <element> | -e <eid> }
```

Description

Note: as an alternative to “*cat*”, you can use the Windows-friendly command-name “*type*”.

The *cat* command retrieves the contents of a particular version of a file element from depot storage in the AccuRev repository. Exception: if you omit the *-v* option, it uses the contents of the file in your workspace tree (on the client machine) rather than going to the repository (on the server machine).

cat displays the data — i.e. sends it to the **stdout** device. You can use ordinary command-shell program techniques to pipe or redirect the data.

This command follows element links and symbolic links, displaying the contents of the target element.

Options

-v <ver-spec>

Display a particular version of the element, instead of the file in your workspace tree. See [Using a Specific Version of an Element](#) on page 14 for a description of the forms that <ver-spec> can take.

For an element link, this option specifies a version of the link element. The current content of the element targeted by the specified version of the element link is displayed.

-p <depot-name>

Specify the depot in which the element resides. The option is required only if the current working directory is not within a workspace for that depot, and you use a stream-number with *-v* instead of a stream-name. When using this option, you cannot specify the element with a simple filename; use its depot-relative pathname (or its element-ID) instead.

-e <eid>

Operate on the file element with the specified element-ID. You can use this option instead of specifying the name of an element.

Examples

Display version 3 in stream **bubble_mary** of file **foo.c**:

```
> accurev cat -v bubble_mary/3 foo.c
```

Display the version of file **foo.c** that is currently used by stream **shell_john**:

```
> accurev cat -v shell_john foo.c
```

Display the version in stream **gizmo_dvt_derek** of the file whose element-ID is **4056**:

```
> accurev cat -v gizmo_dvt_derek -e 4056
```

Display the content of the element link **mylink** as of version 6/2:

```
> accurev cat -v 6/2 -e mylink
```

If version 6/2 of **mylink** pointed to a file **comment.txt** that contained `//10-Jan-2013` when the link was created, but the latest version now contains `//10-Nov-2013`, the command returns `//10-Nov-`

cat

2013. Even if **mylink** has a later version pointing to a different file, the command will return the contents of the latest version of the file pointed to by version 6/2.

See Also

pop

chdepot change the properties of a depot

Usage

```
accurev chdepot -p <depot-name> <new-name>
accurev chdepot -p <depot-name> { -ke | -kd } { -Ci | -Cs }
```

Description

The *chdepot* command can change the name of a depot. A depot's base stream has the same name as the depot itself. Accordingly, *chdepot* also renames the depot's base stream, with a *chstream* command. See [Entity Names](#) on page 5 for information on naming depots.

With the *-k* option, *chdepot* changes the setting of the depot's exclusive file locking property. If this property is enabled (*-ke*), all of the depot's workspaces use exclusive file locking. If this property is disabled (*-kd*), each of the depot's workspaces can be set to use, or not use, exclusive file locking. For more on this feature, see [Workspace Options](#) in the *mkws* reference page.

A depot's case-sensitivity (*mkdepot -C*) can be changed with the *-Cs* and *-Ci* options, within the following restrictions:

- You can always change a depot's case from insensitive to sensitive.
- You can only change from sensitive to insensitive if the depot contains no elements.

Reusing a Depot Name

chdepot does *not* make the depot's original name available for reuse. The only way to reuse a depot name is to

- Rename the depot using *chdepot*.
- Use the AccuRev administration utility command *maintain rmdepot* to completely remove the depot from the AccuRev repository. See [Removing a Depot from the AccuRev Repository](#) on page 122 of the *AccuRev Administrator's Guide*.
- Delete or rename the depot's directory in the AccuRev **storage** directory tree
- Create a new depot with the original name using *mkdepot*.

Options

-p <depot-name>

Specify the depot to be renamed.

-ke

Enable exclusive file locking on a depot-wide basis. All of the depot's workspaces use exclusive file locking.

-kd

(default) Disable depot-wide exclusive file locking. Each of the depot's workspaces can be set individually, either to use or not to use exclusive file locking.

-Cs

Set the depot to be case sensitive.

chdepot

-Ci

Set the depot to be case insensitive. You can do this only if the depot contains no elements. This is useful when you first set up a depot with *mkdepot* and realize, before using it, that it should have been set to be insensitive.

Examples

Fix a misspelled depot name:

```
> accurev chdepot -p mercy mercury
```

Force all of a depot's workspaces to use exclusive file locking:

```
> accurev chdepot -p mercury -ke
```

Change a depot to be case sensitive:

```
> accurev chdepot -p mercury -Cs
```

Change a depot to be case insensitive (will fail if the depot is not empty):

```
> accurev chdepot -p mercury -Ci
```

See Also

chstream, *mkdepot*, *show* (depots)

chgroup

chgroup

rename a group

Usage

```
accurev chgroup <group-name> <new-name>
```

Description

The *chgroup* command changes the name of a user group. See *Entity Names* on page 5 for information on naming groups.

You can subsequently create a new group with the original name, or rename an existing group to the original name. AccuRev will consider the two groups to be distinct, even though they've shared the same name (at different times).

Examples

Fix a misspelled group name:

```
> accurev chgroup raingers rangers
```

See Also

addmember, *mkgroup*, *show* (groups)

chmod change the access mode of an element (UNIX-specific)

Usage

```
accurev chmod { ugo+x | ugo-x } [ -c <comment> ] <element-list>
```

Description

Note: you can execute this command on any AccuRev client machine, but it affects the access rights only on UNIX/Linux systems, not on Windows systems.

The *chmod* command changes the UNIX/Linux access mode of the specified elements. It creates a new version of each element, with all the executable bits (user, group, other) either set or cleared. There is no way to control the executable bits individually.

Options

-c <comment>

Specify a comment for the transaction. The next command-line argument should be a quoted string. Alternatively, the next argument can be in the form **@<comment-file>**, which uses the contents of text-file *<comment-file>* as the comment.

Examples

Set the executable bits on two files:

```
> accurev chmod ugo+x gizmo.c base.h
```

Clear the executable bits on all files with a .c suffix:

```
> accurev chmod ugo-x *.c
```

See Also

add

Techniques for Selecting Elements on page 10

chpasswd change the password of a user

Usage

```
accurev chpasswd [ <name> [ <new-password> ] ]
```

Description

The *chpasswd* command changes the AccuRev password of a registered AccuRev user. Both *<name>* and *<new-password>* are part of AccuRev's security system, and are stored in the AccuRev repository. Don't confuse your AccuRev principal-name and password with the username and password maintained by the operating system. (The names and passwords can match, but they are two distinct data items.)

AccuRev allows you to change any user's password, not just your own. We strongly recommend that your organization use a *server_admin_trig* trigger to provide an appropriate level of control over the ability to change user passwords.

There are several ways to use this command:

- If you enter the command *accurev chpasswd* with no additional arguments, AccuRev changes your own password — it prompts you to enter a new password, and then to confirm by entering it again. The password characters are echoed as asterisks (*).
- If you specify a username — yours or someone else's — on the command line after *chpasswd*, AccuRev changes that user's password. As above, it prompts you to type the new password twice.
- You can specify both a username and a new password on the command line. This eliminates the double prompting for the new password. But there's a potential security hole: if your command shell process has command logging enabled (using environment variable `ACCUREV_COMMAND_LOGFILE`), the new password will appear in the log file, since it is on the command line.

Password Storage and Password-Change Procedure — AccuRev Login Scheme

Your password is stored in the AccuRev repository in encrypted format.

Examples

Change the password of user **john_smith** to **verYtas**:

```
> accurev chpasswd john_smith verYtas
```

Change your password, letting *chpasswd* prompt you to enter and confirm the new password:

```
> accurev chpasswd
New password: *****
Confirmation: *****
Changed password for user alex
```

See Also

login, *logout*, *mktrig*, *mkuser*, *remove* (user), *show* (users)

chref change the name and/or definition of a reference tree

Usage

```
accurev chref -r <reftree-name> <new-name>

accurev chref -r <reftree-name> [ -l <new-location> ] [ -m <new-machine> ]
[ -e <eol-type> ]
```

Description

Note: Before changing the location of a reference tree, consult [A Word of Caution on Windows Zip Utilities](#) on page 11 of the *AccuRev Administrator's Guide*.

The **chref** command registers with AccuRev the fact that a reference tree has changed location. However, **chref** does not physically move the contents of the reference tree. Before using **chref**, you must move the reference tree yourself, using operating system commands such as **tar** (UNIX/Linux) or **xcopy** (Windows).

You can also use **chref** to change the name of a reference tree. You can subsequently create a new reference tree with the original name, or rename an existing reference tree to the original name. AccuRev will consider the two reference trees to be distinct, even though they've shared the same name (at different times). See [Entity Names](#) on page 5 for information on naming reference trees.

Note: You cannot change the backing stream of an existing reference tree. You must use **mkref** to create a new reference tree.

Options

-r <reftree-name>

Specify the name of the reference tree to be changed.

-l <new-location>

Specify the pathname where you have moved the reference tree. You must use a pathname that is valid on the machine where you are executing the **chref** command; if necessary, AccuRev will figure out the actual pathname on the machine where the reference tree has been moved.

Typically, you **cd** to the location where you have moved the reference tree, then use "." as the argument to the **-l** option. See [Examples](#) below.

-m <new-machine>

Specify the hostname where you have moved the reference tree. It is usually not necessary to use this option, even if you move the reference tree to another machine.

-e <eol-type>

Specify the line terminator to be used when an AccuRev command (e.g. **update**, **co**) copies a version of a text file from depot storage to the reference tree. (In depot storage, all text files are stored with the AccuRev-standard line terminator, NL.) The **<eol-type>** can be **d** (default: use the standard line terminator for the client machine's operating system), **u** (use the UNIX/Linux line terminator, NL), or **w** (use the Windows line terminator, CR-NL).

chref

Examples

Rename a reference tree:

```
> accurev chref -r nbuild nightly_build
```

After moving the contents of reference tree **amber_1.3.2** to a network storage area accessed on your machine as **/net/bigdisk/reftree/amber132**, register the new reference tree location:

```
> accurev chws -w amber_1.3.2 -l /net/bigdisk/reftree/amber132
```

See Also

mkref, *show* (*refs*)

chslice

change the location of a slice

Usage

```
accurev chslice -s <slice-number> -l <new-location>
```

Description

Note: Before changing the location of a slice, consult [A Word of Caution on Windows Zip Utilities](#) on page 11 of the *AccuRev Administrator's Guide*.

The **chslice** command registers with AccuRev the fact that a slice of the AccuRev repository has changed location. However, **chslice** does not physically move the slice. Before using **chslice**, you must move the slice yourself, using operating system commands such as **tar** (UNIX/Linux) or **xcopy** (Windows). See [Entity Names](#) on page 5 for information on slice pathnames.

Using 'chslice' with a Replicated Repository

In a replication environment, there are several instances of the same repository — a single master repository, along with one or more replica repositories. Each instance of the repository has its own set of depot slice locations. A given **chslice** command is processed by the AccuRev Server process on a particular machine, and the command affects only the repository instance on that machine.

The locations of a repository's depot slices are stored in the **slices** table in the **site** schema of the AccuRev database.

See [Replication of the AccuRev Repository](#) on page 43 on the *AccuRev Administrator's Guide*.

Options

-s <slice>

Specify the number of the slice to be changed. Use the **show depots** command to list the repository's depots and the corresponding slice numbers. Use the **show slices** command to list the current locations of the repository's slices.

-l <new-location>

Specify the pathname where you have moved the slice. This must be an absolute pathname, local to the AccuRev server machine. It cannot be a network pathname, such as a Windows UNC pathname. Be sure to enclose the pathname in quotes if it contains a SPACE character (such as C:\Program Files\accurev_storage\...).

Examples

Find out the slice number of the **gizmo** depot:

```
> accurev show depots
Depot      Depot#    Slice#
...
gizmo      14        14
...
```

Change the location of the **gizmo** depot's slice:

```
> accurev chslice -s 14 -l /u1/ac_slices/gizmo
```

`chslice`

See Also

[mkdepot](#), *[show](#)* (depots and slices)

chstream

change a stream

Usage

```
accurev chstream -s <stream> <new-name>

accurev chstream -s <stream> [ -b <new-backing-stream> ]
    [ -t <time-spec> ]

accurev chstream -s <stream> [ -b <new-backing-stream> ]
    [ -t <time-spec> ] <new-name>
```

Description

The *chstream* command can change any of these specifications of an existing stream:

- the name of the stream (see [Entity Names](#) on page 5 for information on naming streams)
- the basis time of the stream
- the backing stream: the other stream on which the stream is based

Such changes apply only from the current time forward — it is impossible to change the past. Thus, queries that refer to the past will use the old characteristics of the stream.

Note: Although *chstream* enables you to give a new name to a stream, you cannot then create a new stream with the old name. The old name remains associated with its stream. The only way to reuse a stream name is to completely remove the stream's depot from the AccuRev repository, using the AccuRev administration utility, *maintain*. This utility is documented in the *AccuRev Administrator's Guide*.

Options

-s <stream>

(required) Specify the stream to be changed.

-b <backing-stream>

Specify a new backing stream for the stream to be changed.

-t <time-spec>

Specify a new basis time for the stream. A time-spec can be any of the following:

- Time in <"YYYY/MM/DD HH:MM:SS"> format, "2007/08/07 20:27:15", for example. Note that you must use quotes when specifying time-spec as YYYY/MM/DD HH:MM:SS to ensure that the entire argument is interpreted by the command shell as a single token.
- Time keyword: **now**
- Transaction number as a positive integer: e.g. **146** or **23965**
- Transaction number keyword: **highest**

Use the time keyword **none** to remove a stream's basis time. You cannot set a basis time for a pass-through stream.

When using the **-t** option, take into account the possibility of a timewarp. The AC_SYNC environment variable (see [AccuRev User Preferences](#) on page 6) determines how a timewarp will

chstream

be handled. See also section [System Clock Synchronization](#) on page 31 of the *AccuRev Administrator's Guide*.

Examples

Rename a stream:

```
> accurev chstream -s tulip_dvt tulip_development
```

Change the backing stream of stream **emergency_1017**:

```
> accurev chstream -s emergency_1017 -b gizmo_beta
```

See Also

[mkstream](#), [remove](#) (*stream*), [show](#) (*streams*)

chuser

rename or relicense a user

Usage

```
accurev chuser { -kf | -kd | -kk } <principal-name> <new-name>
[ -e <email_address> ] [ -s <ssh_public_key> ]
```

Description

The **chuser** command changes the principal-name of an existing AccuRev user and/or changes the user's AccuRev licensing. A principal name must start with a letter (a-z, A-Z); it can contain only letters, underscores, spaces, hyphens, periods, plus signs, the commercial at sign (@), and numbers.

Don't confuse your AccuRev principal-name and password with the username and password maintained by the operating system. (The names and passwords can match, but they are two distinct data items.)

Note: After changing the principal-name with chuser, that user's workspace(s) will still be appended with the principal-name that was in effect when the workspace was created. To change the workspace name to reflect the new principal-name, use the **chws -s** command. See [chws](#) for details.

Options

-e <email_address>

Allows you to specify an email address for the user. The address length is limited to 256 characters.

-kf ("full") Change the user to be licensed for AccuRev, AccuWork, and GitCentric.

-kd Change the user to be licensed for AccuWork only.

-kk Create a user licensed for both GitCentric and AccuWork.

-s <ssh_public_key>

Allows you to specify an SSH public key for the user. When specifying the **-s** option:

- Enter text characters only. If you are copying the SSH key from an **id-rsa.pub** file, for example, make sure not to introduce new-line characters or spaces.
- You can omit the **ssh-rsa** string that appears before the key, and the email address that appears at the end of the key. If you choose to include these values, however, the entire argument for the **-s** option must be in double quotes. For example:

```
"ssh-rsa AAAAB3NzaC1yc2EAAAADA.../tvfvdf017me/wh user@company.com"
```

Note that ssh-rsa is assumed. If you are using ssh-dss, you must include the entire argument as shown in the preceding example, substituting **ssh-dss** for **ssh-rsa**.

A general discussion about the SSH protocol is beyond the scope of this document, so use a search engine with terms like 'SSH', and 'tutorial' for more information.

Examples

Rename a user:

```
> accurev chuser john_smith johnny
```

chuser

Change a user so that he can use AccuWork only.

```
> accurev chuser -kd kevin
```

See Also

mkuser, *remove* (*user*), *show* (*users*)

chws change the name and/or definition of a workspace

Usage

```
accurev chws -w <workspace> <new-name>

accurev chws -w <workspace> [ -l <new-location> ] [ -m <new-machine> ]
    [ -b <new-backing-stream> ] [ -k <kind> ] [ -e <eol-type> ]

accurev chws -s <workspace> ...
```

Description

Note 1: before changing the location of a workspace, consult [A Word of Caution on Windows Zip Utilities](#) on page 11 of the *AccuRev Administrator's Guide*.

Note 2: A workspace that currently uses the exclusive-file-locking feature cannot subsequently be changed with the **chws** command. This restriction applies to workspaces whose depot uses the exclusive-file-locking feature.

The **chws** command can change any of the following specifications of an existing workspace:

- name of the workspace (see [Entity Names](#) on page 5 for information on naming workspaces)
- kind of workspace
- location of the workspace tree (hostname and/or pathname)
- the backing stream of the workspace

If appropriate, the **chws** command registers with AccuRev the fact that a workspace has changed location. However, **chws** does not physically move the contents of the workspace. Before using **chws**, you must move the workspace's contents yourself, using operating system commands such as **tar** (UNIX/Linux) or **xcopy** (Windows).

Note: although **chws** enables you to give a new name to a workspace, you cannot then create a new workspace with the old name. The old name remains irrevocably associated with its workspace.

Options

-w <workspace>

Specify the name of the workspace to be changed. You don't have to include the `_principal-name>` suffix.

-s <workspace>

Specify the name of the workspace to be changed. Use the **-s** option when you change a workspace that belongs to another user. In this case, you must include the `_principal-name>` suffix, in order to fully specify the workspace name. This is also useful when you have used the **chuser** command to change the principal-name of a user. Use **chws -s** to update this user's workspace(s) to reflect the new principal-name.

<new-name>

Specify a new name for the workspace. This also renames the workspace stream.

chws

With the **-w** option, you are restricted to changing the name of one of your own workspaces. **chws** adds the suffix `<your-principal-name>` to the new name (if you don't include the suffix yourself).

With the **-s** option, you can change the name of another user's workspace. **chws** does not add any suffix to the new name, so you must include the `<principal-name>` suffix yourself.

See [Entity Names](#) on page 5.

-k *<kind>*

Change the kind of workspace. See the [mkws](#) reference page for details on the specifiers **d** (default storage scheme), **e** (exclusive file locking), and **a** (anchor required).

-b *<new-backing-stream>*

Specify the stream to become the new backing stream of the workspace.

Note: after “reparenting” a workspace to a new backing stream, use the **update** command in that workspace to make sure it contains the correct version of each element.

-l *<new-location>*

Specify the pathname where you have moved the workspace tree. You must use a pathname that is valid on the machine where you are executing the **chws** command; if necessary, AccuRev will figure out the actual pathname on the machine where the workspace tree has been moved.

Typically, you **cd** to the location where you have moved the workspace tree, then use “.” as the argument to the **-l** option. See [Examples](#) below.

-m *<new-machine>*

Specify the hostname where you have moved the workspace tree. It is usually not necessary to use this option, even if you move the workspace tree to another machine.

-e *<eol-type>*

Specify the line terminator to be used when an AccuRev command (e.g. **update**, **co**) copies a version of a text file from depot storage to the workspace. (In depot storage, all text files are stored with the AccuRev-standard line terminator, NL.) The *<eol-type>* can be **d** (default: use the standard line terminator for the client machine's operating system), **u** (use the UNIX/Linux line terminator, NL), or **w** (use the Windows line terminator, CR-NL).

Examples

After moving the contents of your workspace to a network storage area accessed on your machine as **/net/bigdisk/wks/mary/amber_devel**, register the new workspace location:

```
> accurev chws -w amber_dvt_mary -l /net/bigdisk/wks/mary/amber_devel
```

Reparent your workspace to a new development stream:

```
> accurev chws -w amber_dvt_mary -b q2_project_dev
```

After using **chuser** to correct a user's principal-name from “fredriks” to “frederick”, change the user's “mercury_itr” workspace to reflect the corrected principal-name.

```
> accurev chws -s mercury_itr_fredriks mercury_itr_frederick
```

See Also

[excl](#), [incl](#), [mkref](#), [mkws](#), [show](#) (refs and wspaces)

clear

clear remove an include/exclude rule

Usage

```
accurev clear [ -s <stream> ] [ -fx ] { -l <list-file> | <existing-rule> }
```

Description

The **clear** command removes an existing include/exclude rule from the specified stream. If you don't specify a stream, the command applies to the workspace containing the current working directory. You must specify the existing rule with a depot-relative pathname, just as it appears in an *lsrules* listing.

When you remove a rule from a stream, the effect is immediate on the stream itself and on streams below it. The effect does not take place on workspaces below the stream until they are *Update*'d.

When you remove a rule from a workspace, the effect is immediate on the workspace itself: files are copied into the workspace tree if you remove an exclude rule; files are deleted from the workspace tree if you remove an include rule.

Options

-s <stream>

The name of the stream in which the include or exclude rule was explicitly set. (Use *lsrules -fx* to determine this information.) If you use this option, you must specify the existing rule with a depot-relative pathname.

-fx Display the results in XML format.

-l <list-file>

Process the include/exclude rules listed in <list-file>. This must be a text file, with one rule per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the rules. There is no provision for comment lines in a <list-file>.

If you use this option, you cannot also specify an <existing-rule>.

<existing-rule>

An existing include/exclude rule. If you specify a rule in this way, you cannot also use the **-l** option.

Examples

Remove the rule for subdirectory **perl** from the current workspace:

```
> accurev clear \.\tools\perl
```

Remove the rule for subdirectory **perl** from stream **kestrel_test**:

```
> accurev clear -s kestrel_test \.\tools\perl
```

See Also

excl, incl, lsrules, update

co (check out) add an element to the default group of a workspace

Usage

```
accurev co [ -c <comment> ] [ -n [ -O ] ] [ -9 ] [ -R ]
           [ -v <ver-spec> ]
           { -l <list-file> | <element-list> | -e <eid> | -t <transaction-number> }
```

Description

The **co** (“check out”) command adds elements to your workspace’s default group, the collection of elements that are under active development in that workspace. With the **-v** or **-t** option, it also overwrites the file in your workspace with an old version of the element. Be careful when using these options:

- It might overwrite your only copy of a file that you have modified but not yet kept.
- Typically, you’ll need to perform a *merge* before you can promote the file.

If any element is already in the default group (e.g. you have kept one of the files), the **co** transaction is aborted.

An important effect of the **co** command is to “shield” the specified elements from being changed by an *update* command. *update* always skips over the members of the default group when deciding which elements to update.

Exclusive File Locking and Anchor-Required Workspaces

See [File Locking in Workspaces](#) on page 5.

Removing Elements from the Default Group

To remove an element from the workspace’s default group, use the *purge* command. This also overwrites the file in your workspace with the version in the backing stream.

Timestamps on Copies of Old Versions

The **-v** and **-t** options cause an old version of the element to be copied into your workspace. By default, each file copied into your workspace by this command has its timestamp set to the current time. If environment variable `ACCUREV_USE_MOD_TIME` is set, the timestamp is set to the time the version was originally created. (Note: that’s the time the version was created in some user’s workspace stream, not the time it was subsequently promoted to the backing stream.)

Options

-c <comment>

Specify a comment for the transaction. The next command-line argument should be a quoted string. Alternatively, the next argument can be in the form `@<comment-file>`, which uses the contents of text-file `<comment-file>` as the comment.

- O** When used with the **-n** option, overrides timestamp optimization. When **-O** is used, *all* files are considered, even those whose modification times have *not* changed since the last time the workspace was updated or searched for modified files. Having to check all files, regardless of modification time, slows **co** performance. In anchor-required workspaces, overrides the warning message encountered when operating on an element that is active in a sibling workspace.

co

- 9** Add the specified version of the element(s) to the default group, even if a version of the element is in the default group already. Used to “check out” a version which had been previously added or kept but not promoted.
- n** Select all of the modified elements in the workspace that are not already in the default group.
- R** In addition to processing the specified files (if any), recurse into any specified directories and check out all (**modified**) files in those directories that are not already in the default group. (You must also specify **-n**.)
- v <ver-spec>**
Check out the specified version, copying the file to your workspace.
- t <transaction-number>**
Check out all the versions of elements involved in the specified transaction. (You must specify the transaction by number, not by time.) You cannot combine this option with **-v**.
- e <eid>**
Operate on the element with the specified element-ID. You can use this option instead of specifying the name of an element. If you also specify a *<list-file>* or *<element-list>*, it is ignored.
- l <list-file>**
Process the elements listed in *<list-file>*. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any *<element-list>* specified is silently ignored.
- <element-list>**
One or more element names, separated by whitespace. If you also specify a *<list-file>* using the **-l** option, this *<element-list>* is silently ignored.

Examples

Add to the default group all files in the workspace that have been modified but are not yet in the default group:

```
> accurev co -n
```

Get and checkout the version of file **foo.c** that is currently used by stream **gizmo_bugfix**:

```
> accurev co -v gizmo_bugfix foo.c
```

Get and checkout a specific previous version of file **foo.c**, which was created in **jsmith**'s workspace:

```
> accurev co -v gizmo_jsmith/5 foo.c
```

See Also

[Using a Specific Version of an Element](#) on page 14.

cpkadd add an entry to a change package

Usage

```
accurev cpkadd [ -p <depot-name> ] [-I <issue-number> [-3] ][ -fx ]
  { -v <ver-streamspec> <element-list> |
  [ -j <ver-fullspec> ] -v <ver-fullspec> -e <eid>
  [ --from_stream=streamname ] | -k | [ -Fx ] -l <list-file> }
```

Description

The *cpkadd* command creates or updates an entry for one or more elements in the specified *change package*. You can specify an element by its name or its element-ID.

Options

-I <issue-number>

The issue record whose change package will be processed.

-3

Specifies that the <issue-number> specified by the **-I** switch is a third-party ITS key rather than an AccuWork issue number.

-k Process all (**kept**) elements in the workspace.

-e <eid>

The element whose change is to be recorded. You must also specify a version with **-v**. If you also specify a <list-file> or <element-list>, it is ignored.

--from_stream=<stream_name>

Displays output all segments of visible variants of the specified issue in the specified stream. For discussions about segments and variants, see the "Change Packages" chapter of the AccuRev *On-Line Help*.

-Fx Signals that <list-file> (see the **-l** option) is an XML-format file, not a flat text file. Example:

```
<elements>
  <e l=".\\file_0001" v="4/1" eid="2"></e>
  <e l=".\\file_0002" v="4/1"></e>
  <e eid="4" v="4/1"></e>
</elements>
```

-j <ver-fullspec>

Specify the basis version. A full version spec includes both stream and version, such as "16/7" (for "stream #16, version #7). Must be used with **-v** <ver-fullspec>.

-l <list-file>

Process the elements listed in <list-file>. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

cpkadd

If you use this option, any *<element-list>* specified is silently ignored.

<element-list>

One or more element names, separated by whitespace. If you also specify a *<list-file>* using the `-l` option, this *<element-list>* is silently ignored.

`-v <ver-spec>` (or `-v <ver-fullspec>`)

(default: the version currently in your workspace) The head version of the change. A full version spec, as required when used with "`-e <eid>`", includes both stream and version, such as "16/7" (for "stream #16, version #7). When used with an "*<element-list>*", rather than "`-e <eid>`", it makes more sense to specify only the stream (such as "`-v 16`", for the latest version of all the elements in the stream).

Unless `-j` is specified, AccuRev automatically determines the basis version of the change. See the [patch](#) reference page for a description of the basis version.

`-p <depot-name>`

The depot in which the issue record is located (default: the depot of the current workspace).

`-fx` Display the results in XML format. If the command succeeds, the response is:

```
<acResponse/>
```

If the command fails, the response is:

```
<acResponse>
  <Message>Issue not found.</Message>
</acResponse>
```

Examples

Create or update a change package entry for element **brass.h** in issue record #982. The head version of the change is the version of **brass.h** currently in the workspace.

```
> accurev cpkadd -I 982 brass.h
```

Create or update a change package entry in issue record #4871, for the element with element-ID 477. The head version of the change is the version in workspace **brass_dvt_john**.

```
> accurev cpkadd -I 4871 -e 477 -v brass_dvt_john
```

See Also

[cpkdescribe](#), [cpkdepend](#), [cpkremove](#), [issuediff](#), [issuelist](#), [patch](#)

[Using a Specific Version of an Element](#) on page 14

cpkdepend list the dependencies of a change package

Usage

```
accurev cpkdepend -s <from-stream> [ -S <to-stream> ] [ -fv ]
    [ -I <"issue_number1 issue_number2..."> ] [-3] ] [ -p <depot-name> ] [ -
fx ]
```

Description

The *cpkdepend* command lists the issue records on which the specified issue record(s) depend. It can also produce a dependency listing for an entire depot.

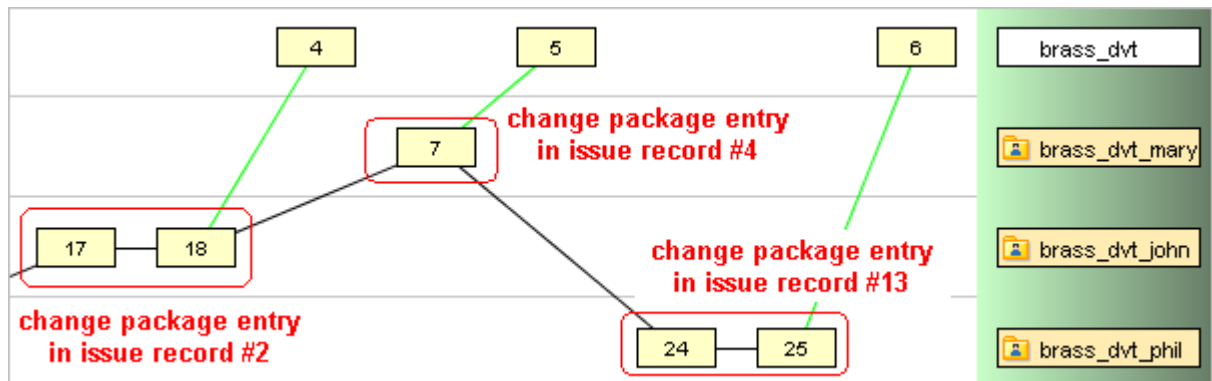
Terminology notes:

- For the purposes of this command, “issue record *N*” is shorthand for “change package of issue record *N*”.
- If issue record A “depends on” issue record B, we say that B “is a dependency of” A.

Change Package Dependencies

The concept of a change package depending on another change package (or an issue record depending on another issue record) is complex, utilizing several aspects of AccuRev’s product architecture. But the basic idea is familiar: when you ask for something, you sometimes get more than you asked for.

This screen shot from the Version Browser in the AccuRev GUI shows a simple example (screenshot not available from the CLI help):



At this point, issue record #13 is active in stream **brass_dvt**. Promoting that issue record (using *promote -Fx -I*) promotes version **brass_dvt/6** (real version: **brass_dvt_phil/25**).

Promoting issue record #13 would give you “more than you asked for”: it would automatically promote issue records #2 and #4, as well. We describe this situation by saying that issue record #13 depends on issue records #2 and #4.

Note: in practice, most issue records’ change packages contain entries for multiple elements, not a single element. In many cases, the promotion causes issue records #2 and #4 to be “partially in” the destination stream, not “completely in”.

The discussion above is conceptual and informal. Here’s a rigorous definition of change package dependency:

cpkdepend

Issue **A** depends on issue **B**, in the context of a promotion from stream **S1** to stream **S2**, if issue **A** is active in stream **S1** and at least one element meets all the following conditions:

- The element has a change package entry in both issue **A** and issue **B**.
- The head version of the element's entry in issue **A** is a descendant of the head version in issue **B**. (That is, the **A** version contains the changes made in the **B** version.)
- Stream **S2**'s version of the element is *not* either the head version of the element's entry in issue **B** or a descendant of it. (That is, the changes made in the **B** version have not yet been promoted to stream **S2**.)

The Change Package Dependency Hierarchy

For each issue record, there is a dependency hierarchy: the issue directly depends on several issues; each of those issues directly depends on several issues; and so on. *cpkdepend* does not display dependencies hierarchically, though. (The AccuRev GUI does.) Instead, it displays one or more lists:

- If you omit the *-fv* option, *cpkdepend* displays a single list: the set of issue records that are direct or indirect dependencies of the issue(s) specified with *-I*:

```
> accurev cpkdepend -I 9 -s brass_dvt
Issue           Depends On
9               7, 1, 3, 2
```

```
> accurev cpkdepend -I 5,9 -s brass_dvt
Issue           Depends On
5, 9           1, 3, 2, 4, 7
```

- If you include the *-fv* option, *cpkdepend* displays a list of the *direct* dependencies for each issue record in the dependency hierarchy of the issue(s) specified with *-I*:

```
> accurev cpkdepend -I 9 -s brass_dvt -fv
Issue           Depends On
9               7, 2
2               1
7               1, 3, 2
1               3
3               1, 2
```

- If you include the *-fv* option but omit the *-I* option, *cpkdepend* lists all the issue records in the depot that have dependencies. For each such issue record, its set of direct and indirect dependencies is included:

```
> accurev cpkdepend -s brass_dvt -fv
Issue           Depends On
1               3, 2
2               1, 3
3               1, 2
4               1, 3, 2
7               1, 3, 2
8               2, 1, 3
9               7, 1, 3, 2
12             11, 10, 4, 1, 3, 2
```

cpkdepend

Note: *cpkdepend* can report that A depends on B *and* B depends on A. This indicates that both issue records have change package entries with the *same* head version for one or more elements.

The change package dependency hierarchy of an issue record can be quite substantial. Even if just one or two elements cause an issue record (A) to depend on just a few other issue records (B,C,D,...), it might be that *other* elements in the dependent issue records (B,C,D,...) produce a large number of further dependencies.

Handling a Change Package Dependency

Suppose that executing *cpkdepend* reveals that dependencies exist in a particular promotion situation — say, issue record #8 depends on issue records #1, #2, and #3. You can handle this situation by propagating *less* data to the destination stream, or by propagating *more* data:

- “Go for less” alternative: instead of promoting issue record #8, use the *patch* command on one or more elements in its change package, in order to include the changes made for issue record #9, but excluded the changes made for issue records #1, #2, and #3.
- “Go for more” alternative: promote issue record #8, and also promote all of its dependencies: issue records #1, #2, and #3. This ensures that the destination stream gets a “matched set” of source versions.

Options

-s <from-stream>

The promote-from stream. You must specify this option.

-S <to-stream>

The promote-to stream. Default: the parent stream of the stream specified with **-s**.

-I <issue_number1 issue_number2...>

The issue record(s) whose change package dependencies are to be listed. Multiple issues can be specified in a space- or comma-delimited list within quotes.

-3

Specifies that the <issue-number(s)> specified by the **-I** switch are third-party ITS keys rather than AccuWork issue numbers.

-p <depot-name>

The depot containing the issues. Default: the depot of the current workspace.

-fv (“verbose”) For each issue record in the dependency hierarchy(s) of the issue record(s) specified with **-I**, display a list of its direct dependencies. Default: display a single list, showing the direct and indirect dependencies of the issue record(s) specified with **-I**.

If you omit **-I**, display a list of the direct and indirect dependencies of the each issue record that has dependencies in the specified stream context.

-fx Display the results in XML format.

cpkdepend

Examples

Display the direct and indirect dependencies of issue records #1004 and #1443, in the context of promotion from stream **widget_dvt** to its parent stream.

```
accurev cpkdepend -I 1004,1443 -s widget_dvt
```

Display the direct dependencies each issue record in the dependency hierarchy of issue record #1004, in the context of promotion from stream **widget_dvt** to stream **widget_mnt**.

```
accurev cpkdepend -fv -I 1004 -s widget_dvt -S widget_mnt
```

See Also

cpkadd, *cpkdescribe*, *cpkremove*, *issuediff*, *issuelist*, *patch*

cpkdescribe list the contents of a change package

Usage

```
accurev cpkdescribe -I <"issue-number1 issue-number2..."> [ -3 ]
    [ -p <depot-name> ] [ -e <eid> ] [ --from_stream=streamname ]
    [ --include_missing_segments ] [ -fx ] [ -t <time-spec> ]
```

Description

The *cpkdescribe* command lists the elements in the change package associated with the issue number you provide. This command returns the same content that is displayed on the Changes tab of the specified AccuWork issue record.

Options

-I <issue-number1 issue_number2...>

The issue record(s) whose change package(s) are to be listed. Multiple issues can be specified in a space- or comma-delimited list within quotes.

-3

Specifies that the <issue-number> specified by the **-I** switch is a third-party ITS key rather than an AccuWork issue number.

-p <depot-name>

The depot in which the issue record is located (default: the depot of the current workspace).

--from_stream=<stream_name>

Displays all segments of visible variants of the specified issue in the specified stream. For discussions about segments and variants, see the "Change Packages" chapter of the *AccuRev On-Line Help*.

--include_missing_segments

Displays missing segments in the change package for the specified stream. Requires the **--from_stream=<stream_name>** option to identify the stream.

For discussions about segments and variants, see the "Change Packages" chapter of the *AccuRev On-Line Help*.

-fx Display the results in XML format.

-e <eid> Specify the element ID (EID) of a single element to be displayed.

-t <time-spec>

Display the elements that existed in the change package as of the specified time. A time-spec can be any of the following:

- Time in <"YYYY/MM/DD HH:MM:SS"> format, "2007/08/07 20:27:15", for example. Note that you must use quotes when specifying time-spec as YYYY/MM/DD HH:MM:SS to ensure that the entire argument is interpreted by the command shell as a single token.
- Time keyword: **now**
- Transaction number as a positive integer: e.g. *146* or *23965*

cpkdescribe

- Transaction number keyword: *highest*

Examples

List the contents of the change package for issue record #4871 in depot **brass**.

```
> accurev cpkdescribe -I 4871 -p brass
```

In XML formatted output, list the information only for the element with an EID of "615", in the change packages for issue records #4871 and #3597 in depot **brass**.

```
> accurev cpkdescribe -I "4871 3597" -e 615 -p brass -fx
```

List all segments of the visible variants associated with issue "39" in stream "prod_5000_devel" of depot "prod_5000":

```
> accurev cpkdescribe -I 39 -p prod_5000 --from_stream=prod_5000_devel
```

See Also

cpkadd, *cpkdepend*, *cpkremove*, *issuediff*, *issuelist*, *patch*

cpkremove remove an entry from a change package

Usage

```
accurev cpkremove [ -p <depot-name> ] [ -I <issue-number> [ -3 ] ] [ -fx ]
    { <element-list> | -e <eid> [ -v <seg_head> -j <seg_basis> ] | -k |
    [ -Fx ] -l <list-file> }
```

Description

The *cpkremove* command removes one or more entries from the specified change package — that is, the Changes tab of the specified AccuWork issue record. You can specify the element by its name or its element-ID, and you can specify a specific segment of an EID.

Options

-I <issue-number>

The issue record whose change package entry is to be removed.

-3

Specifies that the <issue-number> specified by the **-I** switch is a third-party ITS key rather than an AccuWork issue number.

-k Process all (**kept**) elements in the workspace.

-e <eid>

The element whose change entry is to be removed. If you also specify a <list-file> or <element-list>, it is ignored.

-v <seg_head>

The head version of the segment to be removed.

-j <seg_basis>

The basis version of the segment to be removed.

-Fx Signals that <list-file> (see the **-l** option) is an XML-format file, not a flat text file.

```
<elements>
  <e l=". \file_0001" v="4/1" eid="2"></e>
  <e l=". \file_0002" v="4/1"></e>
  <e eid="4" v="4/1"></e>
</elements>
```

-l <list-file>

Process the elements listed in <list-file>. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any <element-list> specified is silently ignored.

cpkremove

<element-list>

One or more element names, separated by whitespace. If you also specify a <list-file> using the **-l** option, this <element-list> is silently ignored.

-p <depot-name>

The depot in which the issue record is located (default: the depot of the current workspace).

-fx Display the results in XML format. If the command succeeds, the response is:

```
<acResponse/>
```

If the command fails, the response is:

```
<acResponse>
  <Message>Issue not found.</Message>
</acResponse>
```

Examples

In issue record #4871 in depot **brass**, remove the change package entry for the element with element-ID 729.

```
> accurev cpkremove -I 4871 -e 729 -p brass
```

In issue record #4871 in depot **brass**, remove the change package entry for the segment with basis version of 16/3 and a head version of 15/9 for the element with element-ID 729.

```
> accurev cpkremove -I 4871 -e 729 -p brass -j 16/3 -v 15/9
```

Perform the same task as above, but by using the "-Fx -l" XML technique:

```
> accurev cpkremove -I 4871 -p brass -Fx -l .\element_list.xml
```

where `element_list.xml` has the following contents:

```
<elements>
  <element eid="729" v="15/9" bv="16/3"></element>
</elements>
```

See Also

[cpkadd](#), [cpkdepend](#), [cpkdescribe](#), [issuediff](#), [issuelist](#), [patch](#)

[Techniques for Selecting Elements](#) on page 10

defunct

remove an element from a stream

Usage

```
accurev defunct [ -c <comment> ]
    { -l <list-file> | <element-list> | -e <eid> }
```

Description

The *defunct* command removes elements from active use in your workspace. That is, for each element you specify, it:

- removes the file from your workspace tree
- marks the element as **(defunct)** in the workspace stream

Since the file is gone, operating system commands such as *ls* (UNIX/Linux) or *dir* (Windows) won't find a defunct element:

```
> dir /b d03.txt
File Not Found
```

But the *stat* and *files* commands will see the defunct element in your working stream:

```
> accurev files d03.txt
.\d03.txt          lemon_dvt_mary\8 (4\8) (defunct) (kept) (member)
```

defunct does not remove an element from the depot altogether. (In fact, *no* operation removes an element — that would violate AccuRev's TimeSafe property.) And *defunct* does not make an element disappear for all users. *defunct* just removes an element from a particular workspace. The element remains visible in other streams and workspaces — at least for the time being (see [Propagating Deactivation with the 'promote' command](#) below).

Exclusive File Locking and Anchor-Required Workspaces

See [File Locking in Workspaces](#) on page 5.

Defuncting the Target of a Link

If you defunct the target of an element link, the link is automatically removed from your workspace. The indicators **(missing)** and **(defunct-target)** are added to the link's status. When you promote the defunct target element to the backing stream, the link's status changes from **(defunct-target)** to **(nonexistent-target)**.

If you defunct the target of a symbolic link, the link is *not* removed from your workspace. The indicator **(missing-target)** is added to the link's status.

Defuncting a Directory

If any of the elements you specify is a directory, *defunct* works recursively: it removes the directory itself and elements under that directory. Only the specified directory itself becomes **(defunct)**; the files and directories below it are simply removed from the workspace tree, but do not become **(defunct)** in the workspace stream.

The precise result depends on whether any elements located below the defuncted directory are active:

- If no element below the defuncted directory is active (in the workspace's default group), the entire directory tree is removed from your workspace tree.

defunct

- If one or more elements below the defuncted directory is active, those elements become stranded. A stranded element is in the default group, but there is no valid pathname to the element in your workspace stream. You can view your workspace’s stranded elements, identified by their element-IDs, with the command *stat -i*.

A stranded file or directory is *not* removed from your workspace tree. But the *stat* or *files* command lists the object as (**external**), since the workspace stream no longer has a valid pathname to the object. See *Elements under a defunct parent* of the *AccuRev Technical Notes*.

CAUTION: In all cases of defuncting a directory, a file below the defuncted directory that you have edited — but never preserved with *keep* — will be removed from the workspace tree. (Such files are not officially “active” in the workspace.) This removes data for which there might be no other copy.

Best Practice for Defuncting an Entire Directory Subtree

Based on the information in the preceding section, it follows that if you wish to defunct a multiple-level directory subtree, you should start at the top, not at the bottom. This eliminates unnecessary work, and it keeps lower-level elements from getting (**stranded**) status.

For example, suppose you wish to defunct the directory subtree starting at **dir01**. But before doing so, you first defunct the lower-level file **dir01/sub02/file03.txt**, and also the lower-level directory **dir01/sub06**. At this point, you have two element with (**defunct**) status:

```
.\dir01\sub02\file03.txt    wdgt_john\1 (16\1) (defunct) (kept) (member)
.\dir01\sub06              wdgt_john\1 (16\1) (defunct) (kept) (member)
```

If you then proceed to defunct directory **dir01**, those two elements will become stranded — the previous *defunct* commands made them “active” in the workspace, but the last *defunct* command caused them to have no pathname in the workspace:

```
.\dir01\sub02\file03.txt e:5 wdgt_john\1 (16\1) (defunct) (member) (stranded)
.\dir00\sub06           e:17 wdgt_john\1 (16\1) (defunct) (member) (stranded)
```

It would have been better not to work at all with the elements lower in the directory tree — simply defunct the highest-level element, **dir01**. This removes **dir01/sub02/file03.txt** and **dir01/sub06** from the workspace (presumably, along with many other elements), but does not cause them to become stranded.

In general, use this procedure for defuncting a directory subtree:

1. Using the *stat -d* command, determine whether any element within the subtree is currently active in the workspace or stream.
2. Resolve the status of any active elements, using *promote* or *purge*.
3. Invoke *defunct* on the top-level directory of the subtree.

Propagating Deactivation with the ‘promote’ command

After *defuncting* an element, you use *promote* to propagate its deactivation to your workspace’s backing stream. (This parallels creating a new version of an element with *keep*, then propagating the version to the backing stream with *promote*.) After promotion, even the *stat* and *files* commands won’t see the defunct element.

Note: The following sequence of operations is consistent with the description above, but the results may be “surprising” if you’re unfamiliar with the *defunct/promote* model:

- Defunct file **foo.c**
- Use a text editor to create a new file named **foo.c**

defunct

- Promote file **foo.c**

The result: element **foo.c** is removed from your workspace stream and from its backing stream, and your workspace contains an external file (not under version control) named **foo.c**.

Reactivation with the 'undefunct' Command

To reactivate a defunct element, use the *undefunct* command. You can do this either before or after you promote the deactivation to the backing stream. (For special cases, see the [undefunct](#) reference page.)

The Past and Future of a Defunct Element

In the future, an element that you have *defuncted* will be removed from all streams to which the change is promoted. When other people update their workspaces (or reference trees) that are backed by those streams, the element will be removed.

TimeSafe-ness means that you cannot change the past. This means that a defunct element remains in old snapshots of your stream. After the change is promoted to other streams, the element will remain in old snapshots of those streams, too. You can always get information about the element (if you know which stream it still exists in) using the *hist* command.

Options

-l <list-file>

Process the elements listed in <list-file>. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any <element-list> specified is silently ignored.

<element-list>

One or more element names, separated by whitespace. If you also specify a <list-file> using the **-l** option, this <element-list> is silently ignored.

-c <comment>

Specify a comment for the transaction. The next command-line argument should be a quoted string. Alternatively, the next argument can be in the form **@<comment-file>**, which uses the contents of text-file <comment-file> as the comment.

-e <eid>

The element-ID of the element to be defuncted. If you use this option, you can specify only one element. If you also specify a <list-file> or <element-list>, it is ignored.

Examples

Defunct two files:

```
> accurev defunct old_commands.c old_base.h
```

Defunct each of the elements listed in file **go_away**:

```
> accurev defunct -l go_away
```

Defunct the entire directory tree named **alpha_project**, located at the top level of the depot:

```
> accurev defunct ./alpha_project
```


defunct

See Also

hist, promote, stat, undefunct

Techniques for Selecting Elements on page 10

demote move a version from one stream to a child stream

Usage

```
accurev demote [ -c <comment> ] [ -d ]
  [ -S <to-stream> ] ]
  [ -I <issue-number(s)> [-3] ]
  { -l <list-file> | <element-list> | -t <transaction-number> | -e <eid>
  | -Fx -l <XML-file> }
```

Description

The *demote* command transitions a version from being active in the parent stream to being active in a child stream or workspace. The most common use of *demote* is to back out unwanted changes from a given stream yet still retain the changes in a child stream.

You can demote by file, by issue, and by transaction.

Note: You cannot demote to a destination that has active changes that overlap with the element you are demoting.

Demote and Entity Types

The following table indicates whether it is possible to demote to or from various AccuRev entities.

AccuRev Entity	Demote from?	Demote to?
Dynamic stream	Yes	Yes
Time-based stream	Yes	Yes
Snapshot stream	No	No
Pass-through stream	No	No
Gated stream	No	No
Staging stream	Yes	No
Workspace	No	Yes
Depot (root)	Yes	No
Reference tree	No	No

Note: While you cannot demote an element to or from a pass-through stream, you can demote an element *across* a pass-through stream that exists between the parent stream and a child stream or workspace.

Demote-Related Triggers

You can use the following AccuRev triggers with the *demote* command:

- The *server_preop_trig* trigger fires on the server machine before the *demote* operation takes place, if the trigger has been configured for the depot in which the elements to be demoted reside. You can add a demote command section to this trigger to perform operations on the server before the demote occurs.

demote

- The *pre-demote-trig* trigger fires on the client machine before the *demote* operation takes place. You can use this trigger to perform actions on the client before the demote, such as cancelling the demote based on the user's identity or privileges.
- You can execute the *event-stream* trigger on the server after the *demote* operation completes. This trigger fires when a demote occurs in the parent stream of an event stream. You can use this trigger to perform actions based upon the demote operation, such as automatically starting a build after the demote has taken place.

For more information on triggers, see the [mktrig](#) reference page and [AccuRev Triggers](#) on page 77 of the *AccuRev Administrator's Guide*.

Demote and the Default Group

When you demote an element from a parent stream to a child stream or workspace, the element is removed from the default group of the parent stream, and is added to the default group of the child stream. That is, the element becomes inactive in the parent stream, and active in the child stream.

Options

-c <comment>

Specifies a comment for the transaction. The next command-line argument should be a quoted string. Alternatively, the next argument can be in the form @<comment-file>, which uses the contents of text-file <comment-file> as the comment.

-d

Selects all elements in the default group.

-e <eid>

Operates on the element with the specified element-ID. This is useful for demoting stranded elements from the backing stream. If you also specify a <list-file>, <element-list>, or <transaction-number>, it is ignored.

Tip: Use *stat -i* to list stranded elements.

-I <issue-number(s)>

You can specify the issue records to be demoted as a SPACE-separated list, enclosed in quotes:

```
-I "645 773 812"
```

-3

Specifies that any <issue-number> specified by the -I switch is a third-party ITS key rather than an AccuWork issue number.

-S <to-stream>

Demotes to the specified stream or workspace. The <to-stream> cannot be a snapshot stream, a pass-through stream, a gated stream, or the root of a depot.

-Fx -l <XML-file>

Identifies an XML-format text file that specifies one or more transactions, issue records, or elements you want to demote.

demote

- Example of XML format for transactions:

```
<transactions>
  <id>71</id>
  <id>72</id>
  <id>73</id>
</transactions>
```

- Example of XML format for issues:

```
<issues>
  <id>7</id>
  <id>21</id>
  <id>144</id>
</issues>
```

When specifying issues, you can use an optional **third_party** attribute if you have configured AccuRev to use an ITS other than AccuWork, as in the following example:

```
<issues third_party="true">
  <id>US23432</id>
  <id>DE726</id>
</issues>
```

If the **third_party** attribute is missing or set to “false”, AccuRev assumes that the *<id>* tag content is the AccuWork issue ID. All *<id>* tags must contain the same type of id, either AccuWork or **third_party**. You cannot mix id types in a single input XML file.

- Example of XML format for elements:

```
<elements>
  <e eid="2" location="./file_1"/>
  <e eid="3"/>
  <e eid="6" location="./a/b/file_2"/>
</elements>
```

The attributes allowed for elements are:

- eid = the element id
- location = the depot relative location of the element (this attribute is optional)

-l *<list-file>*

Processes the elements listed in *<list-file>*. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the element names. Wildcards are not expanded. You cannot enter comment lines in a *<list-file>*.

<element-list>

Processes one or more element names, separated by white space. If you also specify a *<list-file>* using the **-l** option, this *<element-list>* is silently ignored.

-t *<transaction-number>*

Demotes all the versions created in the specified transaction.

demote

Note that if you also specify an element or a list of elements, only those elements are demoted from the transaction.

For example, the following command demotes **file_1** from transaction 15:

```
> accurev demote -t 15 file_1
```

Examples

Demote files **commands.c** and **base.h** to test_stream:

```
> accurev demote -S test_stream commands.c base.h
```

Demote the issue records listed in XML file **demote_these_issues.xml** (see the description of **-Fx** above for an example file):

```
> accurev demote -Fx -l demote_these_issues.xml
```

See Also

[*promote*](#)

AccuRev Administrator's Guide

diag

display performance diagnostics

Usage

```
accurev diag
```

Description

The *diag* command displays diagnostic information relating to AccuRev performance, and is useful for finding underpowered or misconfigured hardware.

Note: This command is designed to provide diagnostic information to an AccuRev Customer Support representative.

Here is some example output:

```
Basic CPU : 53927.44
Host name resolution : 3704.36
Memory : 938.96
Guaranteed disk write : 10.64 MBytes/sec
Disk has performance similar to a mounted file system.
Network read : 11.20 MBytes/sec, 11471.51 KBytes/sec
Available network bandwidth is equivalent to typical 100Mbit LAN
Network write : 11.20 MBytes/sec, 11468.77 KBytes/sec
Available network bandwidth is equivalent to typical 100Mbit LAN
```

For all listed results, larger numbers are better than smaller ones. This command tries to characterize certain results (for example, “equivalent to typical 100Mbit LAN”), in order to facilitate troubleshooting.

The results listed under *Basic CPU*, *Host name resolution*, and *Memory* are specific to the *diag* command, and are useful for comparison to a benchmark machine. The *Guaranteed disk write* figure measures the amount of data the disk can write when those writes are guaranteed. Guaranteed writes are used to maintain database integrity when executing commands like *keep* and *promote*.

The *Network read* and *Network write* figures are obtained by sending 20MB over the network. If the reported bandwidth is less than your network supports, there may be network bottlenecks or latency issues to resolve.

See Also

AccuRev Administrator's Guide

diff

diff compare two versions of an element

Usage

Compare a file in your workspace tree (local file system) with another version:

```
accurev diff [ -b | -v <ver-spec> ] [ -cwWIB | -G ] <element-specs>
```

Compare two specified versions:

```
accurev diff -v <ver-spec-1> -V <ver-spec-2> [ -cwWIB | -G ] <element-specs>
```

Compare a version with the previous version:

```
accurev diff [ -v <ver-spec> ] -l [ -cwWIB | -G ] <element-specs>
```

Compare a version with the corresponding basis version:

```
accurev diff [ -b | -v <ver-spec> ] -j [ -cwWIB | -G ] <element-specs>
```

Compare all file versions in two streams:

```
accurev diff -a -i -v <stream-spec-1> -V <stream-spec-2> [ -cwW ]
```

Compare versions in a stream between two transactions:

```
accurev diff -a -i -v <stream-spec> -V <stream-spec>  
[ -t <transaction-1>-<transaction-2> ]
```

Compare versions that have an **(overlap)** status:

```
accurev diff [ --mine | --theirs ] <element-specs>  
accurev diff -v <stream-spec> [ --mine | --theirs ] <element-specs>  
accurev diff -a [ --mine | --theirs ]  
accurev diff -a -v <stream-spec> [ --mine | --theirs ]
```

At the end of any *diff* command, you can add:

```
-- <diff-utility-flags>
```

These flags are the options to be passed through to the program that performs the actual file comparison. They can be specified on the command line, or stored in the ACCUREV_DIFF_FLAGS environment variable.

Description

The *diff* command compares two versions of a file element, or a set of file elements. For certain forms of this command, the meaning of “version” is expanded to include a modified file that you have not yet preserved with the *keep* command. For such a file, there is no official version object in the AccuRev repository.

In its simplest form ...

```
accurev diff myfile.c
```

... *diff* shows the changes you’ve made recently to a file — changes you’ve made since an *update* of the workspace or a *keep* of the file. More precisely, the default is to compare the file in your workspace tree (in the local file system) with the repository version in your workspace stream, as listed by the *stat* or *files* command:

- the active version in the workspace stream, if the element is active in your workspace

diff

- the version inherited from the backing stream or a higher-level stream, if the element is not active in your workspace

Finding the Differences between Two Streams

The command *diff -a -i -v -V* is a very powerful tool for finding the differences between two streams. It processes just the elements that have different versions in the two streams. It indicates the “before” and “after” version-IDs, along with the content changes and namespace changes between the two versions.

The *mergelist* command also shows the differences between two streams; but it doesn’t include the changes in the “source” stream that have already been propagated to the “destination” stream. See the [mergelist](#) reference page for a fuller explanation.

User Preferences

- **AC_DIFF_CLI**: An environment variable used by *diff* to invoke an alternative file-comparison program. (The default file-comparison program is *acdiff* in the AccuRev **bin** directory.) The command string should include these substitution patterns:

%1% or **%1** — filename of the first version to be compared

%2% or **%2** — filename of the second version to be compared

%3% or **%3** — title to be displayed for the first version

%4% or **%4** — title to be displayed for the second version

The values that AccuRev substitutes for the **%3%** and **%4%** patterns are *not* enclosed in double-quotes, even though the values typically include SPACE characters. The values that AccuRev substitutes for the **%1%** and **%2%** patterns *are* enclosed in double-quotes.

Make sure that the file-comparison program you specify is on your search path.

- **ACCUREV_DIFF_FLAGS**: Stores the options to pass through to the file-comparison program used by the *diff* command. If you use *-* to specify these options while issuing the *diff* command, this environment variable is ignored.

Options

Specifying the Elements — <element-specs>

You can use a combination of exact filenames, pathname patterns (wildcards), and filters to specify the set of file elements whose versions are to be compared. *diff* automatically ignores any elements that are not text files.

Some examples of filenames and/or wildcards:

`brass.c brass.h`

Two files in the current directory.

`*.c`

All files in the current directory whose names end with `.c`.

`*.c ./include/*.h`

All files in the current directory whose names end with `.c`, along with all files in subdirectory *include* of the depot’s top-level directory whose names end with `.h`.

`intro.doc chap???.doc`

diff

File **intro.doc** along with all files in the current directory with names such as **chap02.doc** or **chap17.doc**.

The following options implement filters. Use them to select a subset of the elements in your workspace stream.

- a** Select all elements in the workspace. The output only includes elements for which a difference exists.
 - d** Select only elements in the workspace's default group.
 - k** Select only elements that have been kept in your workspace, but not yet promoted. May not be used with **-m** or **-n**.
 - m** Select only elements that have been modified in your workspace. May not be used with **-k**.
 - n** Select only elements that have been modified in your workspace, and are not in the workspace's default group. May not be used with **-k** or **-d**.
 - o** Select only elements with overlapping changes: you have made changes in your workspace, and changes made in another workspace have already been promoted to the backing stream.
 - p** Select only elements that are pending promotion in your workspace.
- You can use filenames and wildcards in combination with a filter. The filter spec must come first, because it's syntactically a command-line option. For example:

```
-k *.doc
```

... selects all elements in the current directory that have the suffix **.doc** and have (**kept**) status.

(For more information on specifying elements, see *Techniques for Selecting Elements* on page 10.)

Specifying the Versions to Compare

Each of the following options specifies a version to be compared. You can use two options to specify both versions. If you use just one option, your workspace supplies the other version. See the valid-combinations listing below for details.

- b** The version in the backing stream. If an element appears *only* in the workspace stream, not in the workspace's backing stream, the *diff* operation continues and displays the following message when complete:

```
./dir03/sub04/my.new.file created
```

If you use the **-b -v** combination, **-b** indicates the version in the stream, P, that is the parent of the stream specified with **-v**. If **-v** specifies a time-based stream, then *diff* uses the version that was in stream P at the basis time, not the version currently in stream P.

Note: if a text-file element is active in a time-based stream, *accurev diff -b* may produce an incorrect result. In this case, we recommend specifying the backing stream by name using this command variant:

```
> accurev diff -v timebased_stream -V backing_stream foo.c
```

- v <ver-spec>**

A particular version (stream-ID/version-number) or the version currently in a particular stream (stream-ID).

diff

-V <ver-spec2>

Same meaning as **-v**. Use along with **-v** to specify the second version to be compared. If you are comparing a stream between two transactions, the versions used with **-v** and **-V** must be the same.

-t <transaction-1-transaction-2>

A transaction range, used with the **-v** and **-V** options to compare the versions in the stream between the two listed transactions.

-j The basis version corresponding to the other specified version. See the [patch](#) reference page for a description of the basis version.

-1 (“dash-one”, meaning “the previous one”) The version that was the “previous occupant” of the stream to which the other specified version belongs. See [Previous Occupant Algorithm](#) in the [anc](#) reference page for a precise explanation of “belongs”.

If you use this option along with **-v**, be sure to specify a complete version-ID — for example, **-v rose_dvt/14**, not **-v rose_dvt**.

Common cases: If the first version was created in a workspace with *keep*, the **-1** option selects the version on which it was based (the predecessor version), which may or may not have been created in the same workspace. If the first version was created in a dynamic stream with *promote*, the **-1** option selects the version that was in the first version’s stream just prior to the promotion.

--mine

Compares the workspace or stream version against the common ancestor in the backing stream. This option identifies the changes in the workspace or stream that did not come from the backing stream; it is the equivalent of diffing against the last version that you merged with.

--theirs

Displays only the changes added to the backing stream since the last time you merged with the backing stream. This option is useful for identifying the changes to the backing stream that caused the overlap condition.

Following are the valid combinations of the above options:

(no options)

Compare the file in the workspace tree with the version in the workspace stream.

-b

Compare the file in the workspace tree with the version in the workspace’s backing stream.

-v

Compare the file in the workspace tree with the specified version (or the version in the specified stream).

-j

Compare the version in the workspace stream with the corresponding basis version.

-b -j

Compare the version in the workspace’s backing stream with the corresponding basis version.

diff

-v -b

Compare the specified version (**X**) with the version in **X**'s stream's backing stream. (Use **-v -V** instead if comparing elements of type **text** that are active in a time-based stream.)

-v -j

Compare the specified version with the corresponding basis version.

-v -1 (dash-one)

Compare the specified version (**X**) with the version in **X**'s stream's backing stream.

-v -V

Compare the two specified versions.

-v -V -t

Compare versions in the stream between two transactions.

Controlling the Comparison Process

The following options control the way in which the two selected versions of an element are compared as text files.

-B Ignore blank lines in the file comparison.

-c Context diff: show three lines above and below the line in which the difference occurs.

-i Information only: Report the IDs of the two versions, but don't actually compare them. This option is valid only in a command that uses a **-v/-V** combination or a **-v/-b** combination. If an element appears *only* in the workspace stream, not in the workspace's backing stream, this option does not list it.

-I Ignore uppercase/lowercase differences in the file comparison.

-w Ignore whitespace in the file comparison.

-W Ignore changes in the amount of whitespace in the file comparison. But report on newly created whitespace and on whitespace that has been completely removed. For example, **-w** doesn't see the difference between **compareme** and **compare me**, but **-W** does.

-G Use AccuRev's graphical Diff tool to perform the file comparison (any value set in `AC_DIFF_CLI` is ignored).

-- <diff-utility-flags>

Any items on the command line following the double-dash (**--**) are passed through to the program that performs the actual file comparison. Typically, each of these flags also starts with a dash (for example, **-c -i -w**). They can be specified on the command line, or stored in the `ACCUREV_DIFF_FLAGS` environment variable.

Examples

Show the difference between the current contents of file **foo.c** and the most recently kept version:

```
> accurev diff foo.c
```

Show the difference between the current contents of file **foo.c** and the version in the backing stream:

```
> accurev diff -b foo.c
```

diff

Show the difference between the current contents of file **foo.c** and the third version in stream **gizmo**:

```
> accurev diff -v gizmo/3 foo.c
```

What has changed between Release 1.0 and Release 2.0 in the **gizmo** source base?

```
> accurev diff -a -i -v gizmo1.0 -V gizmo2.0
```

In a command shell, specify an alternative file-comparison program, located in a directory on your search path:

```
> export AC_DIFF_CLI="diff %1 %2"      (UNIX Bourne shell)
```

```
> set AC_DIFF_CLI=windiff %1 %2      (Windows)
```

Use the UNIX *diff* command to show the difference between the current contents of file **foo.c** and the most recently kept version:

```
> export AC_DIFF_CLI="diff %1 %2"      (tell AccuRev which diff command to use)
```

```
> accurev diff foo.c -- -i -b          (the -i and -b options tell the UNIX diff command to ignore case differences and space changes)
```

Return Values

diff returns 0 for no differences, 1 for differences, or 2 for an error.

See Also

merge, *mergelist*, *patch*

Using a Specific Version of an Element on page 14

disable_ssl

disable_ssl

turn off SSL security

Usage

```
accurev disable_ssl
```

Description

Removes the Secure Sockets Layer (SSL) certificate from the client to allow unencrypted AccuRev client-server communication. You might need to use this command if SSL has been disabled on the AccuRev server, but the AccuRev client is using a previously accepted SSL certificate. In this situation, the AccuRev server will reject the SSL handshake initiated by the AccuRev client and prompt the user of the AccuRev client to disable SSL (using either the GUI or this CLI command).

See Also

[*enable_ssl, get_certificate*](#)

AccuRev Administrator's Guide

eac1 configure element access control list security

Usage

```
accurev eac1 [-fx] [-R] [-s <stream>] [-n <prin>:<priv> | -i <prin>:<priv>
  | -r <prin>:<priv> | -a <prin>:<priv>] [-c <comment>] [-p <depot-name>]
  { <element-list> | -l <list-file> | -e <eid> }
```

Description

The *eac1* command enables you to configure security settings on individual elements. (For setting security on streams and depots, see the *setacl* and *lsacl* commands.) This gives you the ability to:

- Allow or deny access to all versions of an element, no matter what stream they are in, for a specified user or group. A user that is denied access to an element cannot see or view it.
- Specify who can modify the access to an element. This is a higher privilege than just being allowed access.

You do this by setting and modifying *Access Control Lists (ACLs)* and *Access Control Entries (ACEs)*. An ACL is a list of security protections that applies to an element. An ACE is an entry in an ACL that defines a *principal* and a *privilege*.

A principal can be:

- a user
- a group
- *all*

A privilege can be:

- **Full** - the ability to see and view the element and to modify its ACL.
- **Allow** - the ability to see and view the element, but not modify its ACL.
- **Readonly** - prevents the user from modifying the element when using add, keep, move, defunct, and revert commands, and from modifying the element's ACL.
- **Deny** - the inability to see and view the element or modify its ACL.

An ACL contains zero or more ACEs. An element can have only one ACL assigned to it at any point in time. You specify whether to set, add, or remove an ACE (principal and privilege) to an element, and AccuRev takes care of the ACLs automatically. Note that ACLs cannot be created or modified without an element.

Options

- a Add an ACE to the specified element.
- c Comment for the transaction.
- e Specify an element id.
- fx XML formatted output.

eacl

- i** Displays all the elements that have the privilege for the principal specified in the ACE.
- l** Specify a file with a list of element paths.
- n** Set a new ACL on the specified element(s).
- p** Specify a depot if not in a workspace; used with **-e**.
- r** Remove an ACE from the specified element.
- R** Recurse down the element(s) hierarchy.
- s** Specify a stream for path name lookup when not in a workspace.
- y** Force an ACL change (even if it would remove full access for the user).

<prin> principal(s) — either **all**, **<user>**, or **<group>**

<priv> privilege — either **allow**, **deny**, **full**, or **readonly**

Notes:

- If no privilege changes are specified, then the current EACL list is displayed.
- The **-r** option will fail if no matching ACE is found.
- The **-R** option must be used with either **-n**, **-a** or **-r** (or with no other option, to display the EACLs down the hierarchy). When used with no option, **-R** will not display EACLs for elements to which the user is denied access.
- Can only specify one element with the **-R** option.
- The **-c** option can be used to specify a comment for the transaction.
- The **-p** option requires the **-e** option.
- The **-s** option is required when you are not in a workspace and specifying element paths.
- The **-i** option takes only one ACE; you cannot specify individual elements (using **-l**, **-e**, or **.**, for example).
- Use the **hist** command to display the history of ACL changes on an element.
- Use **'-e all'** with **'-p <depot-name>'** to change ACLs on all elements in the depot.
- You can specify multiple ACEs by separating each **<prin>:<priv>** pair with a comma. For example: **all:allow, user1:deny**.
- Only one ACE per principal is allowed on an ACL, so any previous ACE for that principal will be overwritten.
- Specifying **-n** overwrites any existing ACL. If you wish to modify ACL, use **-a** or **-r**.
- User cannot remove **full** access for the current user unless the **-y** option is specified.
- To add an element to the depot and instantly make it inaccessible to other users, while assigning FULL access to the user adding the element, use the **add -d** command. You can then set the EACLs appropriately for elements added in this way.
- If a user is denied access to an element that is part of a change package, they can still promote that change package even though they cannot see the versions that they are denied.

eacl

Examples

Note: For the examples below to work, you must either have FULL rights (not just ALLOW) to the elements, or be an Accurev EACL superuser. See the “AccuRev Security Overview” chapter of the *AccuRev Administrator’s Guide* for more details.

To display the ACL for the specified element(s):

```
eacl { <element-list> | -l <list-file> | -e <eid> }
```

To set an ACL on the specified element(s):

```
eacl -n <prin>:<priv> { <element-list> | -l <list-file> | -e <eid> }
```

To add an ACE to the specified element(s):

```
eacl -a <prin>:<priv> { <element-list> | -l <list-file> | -e <eid> }
```

To remove an ACE from the specified element(s):

```
eacl -r <prin>:<priv> { <element-list> | -l <list-file> | -e <eid> }
```

To set an ACL on the specified directory and all its children:

```
eacl -R -n <prin>:<priv> { <dir-list> | -l <dir-list-file> | -e <dir-eid> }
```

To display all the elements that have the privilege for the principal specified in the ACE, in XML formatted output:

```
eacl -i <prin>:<priv> -fx
```

Note: If you attempt to set or change an element ACL and do not have the appropriate (“FULL”) permission to do so, you will get the error “Full Access Denied: <element_name>”. When attempting to set a “DENY” access on an element, do not misinterpret this message as a confirmation that the “DENY” access has been set. A successful operation returns the message “Processed: <element_name>”.

See Also

[add](#), [lsacl](#), [setacl](#)

[Element-based security](#) in the *AccuRev Administrator’s Guide*.

enable_ssl

accept SSL certificate

Usage

```
accurev enable_ssl [--thumbprint=<thumbprint_value>]
```

Description

In AccuRev installations in which the Secure Sockets Layer (SSL) protocol has been enabled on the AccuRev server, AccuRev prompts the user to enable SSL on the client in order to establish encrypted communication. This is done by inspecting and accepting the SSL certificate provided by the server. One of the ways to do this is to execute the *enable_ssl* command.

The *enable_ssl* command returns information about the SSL certificate and allows you to accept or reject it. If you accept it, AccuRev downloads the certificate to the client machine, enabling you to connect to the AccuRev server. If you reject the SSL certificate, you cannot connect to the AccuRev server.

The *--thumbprint* option allows you to specify the certificate's *thumbprint*, which, if it matches that of the SSL certificate on the AccuRev server, allows the certificate to be accepted automatically.

The *--thumbprint* option can be used to automatically enable SSL on unattended machines using a script that executes

```
accurev enable_ssl --thumbprint=309b7af1445f8b1fac7b6f8baabc3f7bb656dac9
```

for example. (The term thumbprint is the equivalent of *fingerprint*, which might be more familiar to users in UNIX/Linux environments.)

Options

--thumbprint=<thumbprint_value>

Compare the *<thumbprint_value>* with the certificate on the AccuRev server and accept if they match. If they do not, SSL is not enabled and AccuRev returns an error.

The *<thumbprint_value>* is the SHA1 hash of the certificate represented as a hexadecimal string. You can separate hexadecimal characters with colons, or you can double-quote the entire value and separate the characters with spaces. All of the following examples are valid:

```
"30 9b 7a f1 44 5f 8b 1f ac 7b 6f 8b aa bc 3f 7b b6 56 da c9"
```

```
30:9b:7a:f1:44:5f:8b:1f:ac:7b:6f:8b:aa:bc:3f:7b:b6:56:da:c9
```

```
309b7af1445f8b1fac7b6f8baabc3f7bb656dac9
```

Values are case-insensitive, so the following example is valid as well:

```
30:9B:7A:F1:44:5F:8B:1F:AC:7B:6F:8B:AA:BC:3F:7B:B6:56:DA:C9
```

Examples

Visually inspect and accept (or reject) the SSL certificate; enable SSL upon acceptance:

```
> accurev enable_ssl
```

enable_ssl

Compare the SSL thumbprint to that of the SSL certificate on the AccuRev server and automatically accept the certificate and enable SSL if there is a match:

```
> accurev enable_ssl --thumbprint=309b7af1445f8b1fac7b6f8baabc3f7bb656dac9
```

See Also

[disable_ssl](#), [get_certificate](#)

AccuRev Administrator's Guide

excl

excl set a rule to exclude elements from a workspace or stream

Usage

```
accurev excl [ -s <stream> ] [ -fx ] { <element> | -l <list-file> }
```

Description

The *excl* command excludes the specified element (either a directory or a file) from a particular workspace or stream. If you don't use the *-s* option to name a stream, *excl* operates on the workspace containing the current working directory.

Excluding a directory element causes it to disappear from the workspace or stream; all the files and subdirectories below the directory element disappear, too. Excluding a file element causes just that one file to disappear.

Inheritance of Exclusion

When you exclude elements from a workspace, the files are immediately removed from the workspace tree.

When you exclude elements from a higher-level stream, the exclusion instantly applies to streams below it in the depot's stream hierarchy. The exclusion also applies to workspaces below the stream, but the elements don't disappear from a workspace until you *update* it. This inheritance of exclusions down the stream hierarchy stops at snapshots and at dynamic streams whose basis time precedes the time of the exclusion. The element exclusions don't apply to the snapshot (which is, by definition, immutable) or to that time-based stream.

Creating an Empty Workspace Before using 'excl'

Using *mkws* with the *-i* option (to create an initially empty workspace) is a useful technique for organizations that use *incl* and *excl* to control which elements appear in a workspace.

Options

-s <stream>

Apply the exclude rule to the specified stream. If you use this option, you must specify the element using a depot-relative pathname.

-fx Display the results in XML format.

-l <list-file>

Process the elements listed in <list-file>. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in a <list-file>.

If you use this option, any <element-list> specified is silently ignored.

<element>

An element name. If you also specify a <list-file> using the *-l* option, this <element> is silently ignored.

excl

Examples

Exclude the **src** directory tree from the current workspace:

```
> accurev excl \.\src
```

Exclude the **src** directory tree from stream **gizmo_dvt**:

```
> accurev excl -s gizmo_dvt \.\src
```

See Also

clear, incl, incldo, lsrules, update

files

show the status of elements

Usage

```
accurev files [ -s <stream> ] [ -O ] [ -f <format(s)> ] [ --ignore ]
[ -X ] { <element-list> | -l <list-file> }
```

Description

The *files* command lists the AccuRev status of files and directories in a workspace or reference tree.

- For each file element you specify, it lists the element itself.
- For each directory element you specify, it lists the contents of the directory (the file and directory elements within it).
- If you don't specify any elements on the command line, it lists the contents of the current working directory.

As this example shows ...

```
\\.src\brass.c      capon\1 (5\1) (backed)
```

... the listing for each element includes:

- the element's depot-relative pathname
- the virtual version of the element that is currently in your workspace (or in the stream you specified with *-s*)
- the corresponding real version of the element (version-ID in parentheses)
- the status of the element in your workspace or the specified stream

The various status flags — **(backed)**, **(kept)**, etc. — are the same as those displayed by the *stat* command.

Optimized Search for Modified Files

By default, *files* uses a timestamp optimization to speed its search for files that have been modified in your workspace: it does not consider files whose timestamps have not changed since the last time the workspace was updated or searched for modified files.

It *is* possible for your workspace to get new files with old timestamps: certain file-copy and file-archive utilities can preserve timestamps; the AccuRev commands *co*, *pop*, *purge*, *revert*, and *update* preserve timestamps when copying versions into the workspace if the environment variable ACCUREV_USE_MOD_TIME is set. In such situations, the timestamp optimization causes *files* to silently ignore relevant files. Use the *-O* option to have *files* dispense with the optimization and consider all files, regardless of timestamp, in its search for modified files.

Listing of Defunct / Deleted Objects

If you remove a file or directory element from your workspace with the *defunct* command, the *files* command continues to “see” it:

```
\\.src\brass.c      capon_dvt_john\2 (5\2) (defunct) (kept) (member)
```

That's because the element itself is still active in your workspace stream. (You can think of it as actively being changed from present to absent.) The element continues to be listed by *files*, with the **(defunct)** status flag, until you *promote* it or *purge* it. Promoting the defunct element removes it from the workspace

files

stream, and thus from *files* listings; purging the element effectively undoes the *defunct* command, returning the element to **(backed)** status.

Similarly, if you remove a file or directory element from your workspace tree with an operating system command (such as *del* or *rm*), the *files* command continues to “see” it. That’s because the operating system command doesn’t modify the AccuRev element itself. The status of a deleted element in your workspace is **(missing)**.

Note: The *stat* and *files* commands do not include defuncted elements when listing the contents of the parent directory. The operating system’s “list directory” command doesn’t see the defuncted element, because *defunct* removes the file or directory from the workspace tree.

Options

-s <stream>

Display the status of the element(s) in the specified stream, not in your workspace. See the description of <*element-list*> below.

-O Override the optimized the search for modified files. When **-O** is used, *all* files are considered, even those whose modification times have *not* changed since the last time the workspace was updated or searched for modified files. Having to check all files, regardless of modification time, slows *files* performance. See [Optimized Search for Modified Files](#) above.

-f <*format(s)*> Use one or more of the following format letters:

i: Select all files and directories, even if they are excluded by the value of the **.acignore** file. (See the **-x** description, and [Specifying Ignore Patterns for External Objects](#) on page 7 of *AccuRev Technical Notes* .)

I: Select only those files and directories specified by the value of the **.acignore** file.

x: Display the results in XML format.

Note: You can combine the *x* letter with either *i* or *I* (**-fxi**, for example), but the *i* and *I* letters are mutually exclusive (**-fil** is invalid, for example).

--ignore

Specify a pattern of elements; external elements matching this pattern are ignored by the **files** command. The **--ignore** option can be used multiple times for the same command, but each instance can specify only one pattern. For example:

```
accurev files -x --ignore="*.obj" --ignore="*.txt" --ignore="!file1.txt"
```

Can also be used to specify the pathname of a text file that contains one or more patterns. For example:

```
accurev files -x --ignore="@c:\work\my_ignore.txt"
```

See [Specifying Ignore Patterns for External Objects](#) on page 7 of *AccuRev Technical Notes* for more information on specifying ignore patterns for external files.

Note that the patterns expressed using the **--ignore** option take precedence over patterns expressed using the **.acignore** file.

-X Add excluded elements to the listing.

files

-l <list-file>

Process the elements listed in <list-file>. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any <element-list> specified is silently ignored.

<element-list>

One or more element names, separated by whitespace. If you also specify a <list-file> using the **-l** option, this <element-list> is silently ignored.

You can specify patterns, using the wildcard characters **?** (match any character) and ***** (match any 0 or more characters).

If you use **-s** to display the status of elements in a non-workspace stream, then each relative pathname in <element-list> is interpreted relative to the depot's top-level directory. Typically, a simple filename like **base.h** won't be valid; instead, use a pathname like **src/include/base.h**. Alternatively, use a full depot-relative pathname, like **./src/include/base.h**.

Examples

List the status of all files in the current directory:

```
> accurev files
```

```
.\tools\perl\findtags.pl  brass2\1 (2\1) (modified)
.\tools\perl\reporter.pl  brass2_john\2 (2\2) (kept) (member)
.\tools\perl\mail.1      (external)
.\tools\perl\mail.2      (external)
```

List the status of all files in subdirectory **doc**:

```
> accurev files doc
```

```
.\doc\chap01.doc  brass2_dvt\1 (2\2) (backed)
.\doc\chap02.doc  brass2_john\2 (2\2) (kept) (member)
.\doc\chap03.doc  brass2\1 (2\1) (backed)
.\doc\chap04.doc  brass2\1 (2\1) (backed)
```

See Also

stat

Techniques for Selecting Elements on page 10

The Update Algorithm on page 26 of *AccuRev Technical Notes*

get_certificate

get new SSL certificate

Usage

```
accurev get_certificate [--thumbprint=<thumbprint_value>]
```

Description

In AccuRev installations in which the Secure Sockets Layer (SSL) protocol has been enabled on the AccuRev server, AccuRev prompts the user to enable SSL on the client in order to establish encrypted communication. This is done by inspecting and accepting the SSL certificate provided by the server. One of the ways to do this is to execute the *get_certificate* command.

The *get_certificate* command returns information about the SSL certificate and allows you to accept or reject it. If you accept it, AccuRev replaces the certificate on the client machine, enabling you to connect to the AccuRev server. If you reject the SSL certificate, you cannot connect to the AccuRev server.

The *--thumbprint* option allows you to specify the certificate's *thumbprint*, which, if it matches that of the SSL certificate on the AccuRev server, allows the certificate to be accepted automatically.

The *--thumbprint* option can be used to automatically enable SSL on unattended machines using a script that executes

```
accurev get_certificate --thumbprint=309b7af1445f8b1fac7b6f8baabc3f7bb656dac9
```

for example. (The term *thumbprint* is the equivalent of *fingerprint*, which might be more familiar to users in UNIX/Linux environments.)

Options

--thumbprint=<thumbprint_value>

Compare the *<thumbprint_value>* with the certificate on the AccuRev server and accept if they match. If they do not, SSL is not enabled and AccuRev returns an error.

The *<thumbprint_value>* is the SHA1 hash of the certificate represented as a hexadecimal string. You can separate hexadecimal characters with colons, or you can double-quote the entire value and separate the characters with spaces. All of the following examples are valid:

```
"30 9b 7a f1 44 5f 8b 1f ac 7b 6f 8b aa bc 3f 7b b6 56 da c9"
```

```
30:9b:7a:f1:44:5f:8b:1f:ac:7b:6f:8b:aa:bc:3f:7b:b6:56:da:c9
```

```
309b7af1445f8b1fac7b6f8baabc3f7bb656dac9
```

Values are case-insensitive, so the following example is valid as well:

```
30:9B:7A:F1:44:5F:8B:1F:AC:7B:6F:8B:AA:BC:3F:7B:B6:56:DA:C9
```

Examples

Visually inspect and accept (or reject) the SSL certificate; enable SSL upon acceptance:

```
> accurev get_certificate
```


get_certificate

Compare the SSL thumbprint to that of the SSL certificate on the AccuRev server and automatically accept the certificate and enable SSL if there is a match:

```
> accurev get_certificate --  
thumbprint=309b7af1445f8b1fac7b6f8baabc3f7bb656dac9
```

See Also

[disable_ssl](#), [enable_ssl](#)

AccuRev Administrator's Guide

getconfig

list the contents of an AccuWork configuration file

Usage

```
accurev getconfig [ -p <depot-name> ] [ -u <user-name> ] -r <config-file>
    [ -s [ -d <locale> ] ]
```

Description

The *getconfig* command displays the contents of an AccuWork configuration file. Most such files store some of the information defined for a depot in the GUI Schema Editor:

- Contents of the *Schema* subtab: **schema.xml**
- Contents of the *Layout* subtab: **layout.xml**
- Contents of the *Lists* subtab: **lists.xml**
- Contents of the *Relationship Types* subtab: **relation_types.xml**
- Contents of the *Validation* subtab: **logic.xml**
- Contents of the *Workflow* subtab: **custom_actions.xml** and **logic.xml**
- Contents of the Change Package Results section of the *Change Packages* subtab: **cpk_fields.xml**
- Contents of the Change Package Triggers section of the *Change Packages* subtab: **cpk_promote_queries.xml**

These files are stored in subdirectory **dispatch/config** of the depot's storage directory (or slice) within the AccuRev repository.

AccuWork queries, defined on the GUI's Queries tab, are also stored as configuration files:

- A user's AccuWork queries are stored in the depot's storage directory, in file **dispatch/config/user/<username>/query.xml**. (This includes queries that the user has declared to be public.)
- The depot's public AccuWork queries are stored in the depot's storage directory, in file **dispatch/config/publicQuery.xml**.

To support localized default schemas, *getconfig* provides a **-s -d <locale>** option that causes the server to search for a localized site resource based on the specified locale. If a localized resource is installed at the server, it will be returned to the client. If no localized version is available, the default English version of the resource is returned.

Options

-d <locale>

The UNIX-standard locale specification for the localized default schema, in the format:

```
language[_territory][.codeset][@modifier]
```

Note: AccuRev does not currently use any @modifier specifications.

-p <depot-name>

The depot to use (default: the depot of the current workspace).

getconfig

-u <*user-name*>

The AccuRev user whose configuration file is to be displayed. Use this option when displaying a particular user's AccuWork queries (*-r query.xml*).

-r <*config-file*>

The name of the XML-format configuration file.

-s

The specified resource file should be returned from [site_slice/dispatch/config](#).

Examples

Display the AccuWork queries for user **derek**, for the issues in the current depot:

```
accurev getconfig -u derek -r query.xml
```

Display the field validations defined for the issues in depot **widget**:

```
accurev getconfig -p widget -r logic.xml
```

Get the configuration file schema.xml from the Japanese (ja) locale folder of [site_slice/dispatch/config](#):

```
accurev getconfig -s -d ja -r schema.xml
```

Same as above, but with a more detailed locale specification:

```
accurev getconfig -s -d ja_JP.UTF-8 -r schema.xml
```

AccuRev does not currently use any @modifier specifications.

See Also

[putconfig](#)

getpref

list user preferences

Usage

```
accurev getpref
```

Description

The *getpref* command displays your AccuRev user preferences, as stored in file **preferences.xml** in the **.accurev** subdirectory of your AccuRev home directory. This data is in XML format.

(By default, the AccuRev home directory is determined automatically; but you can specify an AccuRev home directory with environment variable ACCUREV_HOME.)

Users Preferences and Environment Variables

If an environment variable has the same name as an entry in the user preferences file, AccuRev commands follow the environment variable and ignore the preference-file entry.

Example

Display your AccuRev user preferences:

```
> accurev getpref
<AcResponse
  Command="getpref">
  <versionMajor>5</versionMajor>
  <versionMinor>0</versionMinor>
  <versionPatch>0</versionPatch>
</AcResponse>
```

See Also

setpref

AccuRev User Preferences on page 6

getproperty list stream or principal properties

Usage

```
accurev getproperty [ -f ] [ -ks |-ku ] [ -p <depot> ] [ -s <stream> |
-u <principal> ] [ -r ] [ -l <file-name> ][ -n <pattern> ]
```

Description

The *getproperty* command displays the name/value pair for the specified stream or principal property. If the property value is a file, *getproperty* displays the file name for the value. If a target file is specified using the **-l** option, *getproperty* writes the file contents to that file.

Options

-f <format(s)>

Specifies output format. Use one or more of the following format letters:

- i**: Include properties for hidden streams or principals.
- v**: Display property name only.
- x**: Display the results in XML format.

-ks | ku

Shows properties for all streams (**-ks**) or all principals (**-ku**). Note that **-ks** requires the use of **-p** to specify the depot.

-p <depot>

Specifies the name of the depot whose properties you want to display.

-s <stream>

Specifies the name of the stream whose properties you want to display.

-u <principal>

Specifies the name of the user or group whose properties you want to display.

-l <file-name>

Specifies the name of the file to which you want the property value written. This option can be used only for a property value that was originally specified as a file (setting a "profile" property to a .png file, for example).

This option creates the file you specify, and the command fails if the file already exists. You can use either a relative or absolute path to specify the file location. If you do not specify a path, AccuRev assumes the file is in the current directory. The file should have an extension appropriate for the data that will be written to it. If the property is an image, the file should have a .png or .jpg extension, for example.

-n <pattern>

Specifies the glob pattern the property name must match; only properties whose name matches the glob pattern are displayed.

The glob pattern is a simple pattern that allows "*" to specify one or more unmatched characters or "?" to match exactly one unmatched character. Use the backslash (\) as an escape character to pass a literal "*" or "?". A name with no pattern characters will be exactly matched.

getproperty

Note: You must use **-n** to specify a pattern for the property name if you use the **-l** option.

-r

Option required to get reserved properties for the specified stream (**-s**), or for all the streams in the specified depot (**-p**).

Examples

Get all properties on stream "prod_2010":

```
> accurev getproperty -s prod_2010
```

Get the value of the "streamStyle" reserved property on stream "nightly":

```
> accurev getproperty -s nightly -r -n streamStyle
```

Get the value of all reserved properties on stream "nightly" and display them using XML format:

```
> accurev getproperty -s nightly -r -fx
```

Get all properties on all streams in depot "phoenix":

```
> accurev getproperty -ks -p phoenix
```

Get the value of the "profile" property on user "jsmith":

```
> accurev getproperty -u jsmith profile
```

Get the value of the "profile" property on user "jsmith" and write the value to "c:\temp\jsmith.txt":

```
> accurev getproperty -u jsmith -l c:\temp\jsmith.txt -n profile
```

Note: The preceding example assumes that the value of the "profile" property was specified as a file.

XML Output Examples

Stream Properties

```
<?xml version="1.0" encoding="utf-8"?>
<properties>
  <property
    kind="stream"
    depotName="test"
    streamNumber="2"
    streamName="prod_2010"
    propertyName="Status">obsolete</property>
  <property
    kind="stream"
    depotName="test"
    streamNumber="2"
    streamName="prod_2010"
    propertyName="Project">Amber</property>
</properties>
```

User Properties

```
<?xml version="1.0" encoding="utf-8"?>
<properties>
  <property
    kind="principal"
    principalNumber="1"
    principalName="jsmith"
    propertyName="profile">*</property>
</properties>
```

See Also

setproperty, *rmproperty*, *show*

help

display help on the AccuRev CLI

Usage

```
accurev help [ -s | -u | -v ] [ <help-category> | <command-name> ]
```

Description

The *help* command displays help text for an individual AccuRev CLI command or summary information on a category of commands. For an individual command, the help text is essentially the same as the command's "reference page" in this manual. The command categories are:

all — list all AccuRev CLI commands, alphabetically
ws — workspace-related commands
vc — basic version-control commands
rpt — reporting commands
resolve — commands for resolving differences between element versions
moving — commands for renaming, moving, or removing elements
security — security-related commands
repl — replication-related commands

Invoking the *help* command without any arguments displays the category list.

Alternative Method for Invoking Help

For any individual command, you can use the *-h* option to display the command's help text. For example, these two commands are equivalent:

```
accurev keep -h
accurev help keep
```

Options

- s** For an individual command, display only the command name, summary description, and command syntax.
- u** Same as **-s**.
- v** For an individual command, display the complete reference page. By default, the DESCRIPTION section is omitted from the display.

Examples

Display a summary of AccuRev's version-control commands:

```
> accurev help vc
```

Display the complete reference page for the *diff* command:

```
> accurev help -v diff
```

hist show the transaction history of elements or an entire depot

Usage

```
accurev hist [ -t <transaction-range> ] [ -s <stream> ] [ -c <comment> ]
  [ -u <users> [ --exclude_users ] ]
  [ -k <transaction-kinds> [ --exclude_kinds ] ]
  [ -f <format(s)> ] [ --show_historic_cpk ]
  { -a | <element-list> | [ -Fx ] -l <list-file> | -e <eid> }
  [ -p <depot-name> ] [ -R ]
```

Description

Note: As an alternative to “*hist*”, you can use the command-name “*history*”.

The *hist* command displays part or all of the transaction history of one or more elements. For each transaction, it reports the creation time, who made changes, comments for each version, etc. For each version involved in a transaction, it reports the virtual version, the real version, the transaction number, the transaction time, who created the version and how, and the comment.

Options

- a** Operate on all elements in the depot. You can use this option instead of specifying a list of elements.
- c <comment-string>**
Display only transaction(s) whose comments contain the specified string. (The string matching is case-insensitive.) Enclose the string in quotes to ensure that *hist* interprets it as a single command-line argument.
- e <eid>**
Operate on the element with the specified element-ID. If you also specify a <list-file> or <element-list>, it is ignored.
- p <depot-name>**
By default, AccuRev uses the depot of the current workspace as its context. Use this option to specify a different depot, or to specify a depot when there is no current workspace.
- R** Recurse into each directory element specified and continue to show transaction history within. The **-R** option requires at least one element or the **-l** option. The **-R** option cannot be used with the **-a** option.
- s <stream>**
Display only the transactions that involved the specified stream. If you use this option, you must specify elements using depot-relative pathnames or element-IDs (**-e** option).
- f <format(s)>**
Use one or more of the following format letters or numbers:
 - 3:** ("third-party") For issue numbers associated with a version, display the third-party ITS key and value for the issue.

hist

-e: (“expanded”) Display more detail for *promote* transactions: information on the transaction(s) in the current workspace (*keep*, *move*, etc.) that recorded the change(s) being promoted.

-i: (“issue”) Display issues related to a version of an element at the time of the specified transaction. If a version is not currently the head of an issue, but was at the time of the transaction, the issue is displayed with an attribute `historic_only=true`. This attribute is not displayed if the version is currently the head and was also the head at the time of the transaction.

-v: (“verbose”) For *keep* and *add* transactions, include the modification time, checksum, file size, data type (text or binary), and depot-relative pathname. *cpk* transactions are listed as add or remove, followed by details of the *cpk add* or *cpk remove* operation.

(You can use **-fev** to display both expanded and verbose information.)

-l: (“list”) For each transaction, show only the path of the element. For *move* transactions, only the destination is shown.

-s: (“status”) If the displayed version of the element is in the default group of the stream specified with **-s** (default: workspace stream), list it with “(**member**)”. This option is intended to be used in combination with **-l**, when you’re examining a transaction listed by *translist*.

-t: (“transaction”) Display just the header line, including the transaction number.

-x: (“XML”) Display results in XML format.

-Fx Signals that `<list-file>` (see the **-l** option) is an XML-format file, not a flat text file. Example:

```
<transactions>
  <id>1</id>
  <id>6</id>
</transactions>
```

-l `<list-file>`

Process the elements listed in `<list-file>`. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any `<element-list>` specified is silently ignored.

`<element-list>`

One or more element names, separated by whitespace. If you also specify a `<list-file>` using the **-l** option, this `<element-list>` is silently ignored.

-k `<transaction-kinds>`

Display only the transactions of the specified kind(s). Valid values are:

**add, archive, chstream, co, defcomp, defunct,
demote_from, demote_to, dispatch, eacl, keep, move,
mkstream, promote, purge, unarchive, undefunct**

You may specify multiple types using the **-k** option by separating the types with commas.

Note: When you use **-k defcomp**, AccuRev returns transactions of the following types, all of which are related to setting include/exclude rules:

incl, excl, incldo, and clear

If there are no transactions of these types associated with the element, AccuRev returns:

No history corresponding to selection.

Note that you cannot use **-k** to filter these transaction types directly.

hist

--exclude_kinds

Use in conjunction with the **-k** option to indicate that the output should display all kinds of transactions *except for* the named kind(s).

-u <users>

Display only the transactions for the specified AccuRev user(s).

You may specify multiple users using the **-u** option by separating the usernames with commas. If any of the names contains a space, place double quotes around the entire list of names.

--exclude_users

Use in conjunction with the **-u** option to indicate that the output should display transactions executed by all users *except for* the named user(s).

--show_historic_cpk[=true]

Display the state of the related change package as it was at the time of the specified transaction. By default, the current state of the change package is shown. The "=true" argument is optional and explicitly specifying it is the same as omitting it. Providing any other string than "true" is the same as omitting **--show_historic_cpk** altogether and will result in the current state of the change package being displayed.

-t <time-spec>[.<count>]

-t <time-spec> - <time-spec>[.<count>]

The first form specifies one transaction, or a specified number of transactions up to (i.e. preceding) and including a particular transaction.

The second form specifies transactions that took place in the specified interval; the optional suffix truncates the listing after the most recent *<count>* transactions. You may need to use quotes in composing the argument following **-t**; the entire argument must be interpreted by the command shell as a single token. The command output depends on the order of the time-specs; list the most recent time-spec first to show the results in order from most recent to least recent. Reverse the order of the time-specs to reverse the order of the output.

A time-spec can be any of the following:

- Time in *<"YYYY/MM/DD HH:MM:SS">* format, "2007/08/07 20:27:15", for example. Note that you must use quotes when specifying time-spec as *YYYY/MM/DD HH:MM:SS* to ensure that the entire argument is interpreted by the command shell as a single token.
- Time keyword: *now*
- Transaction number as a positive integer: e.g. *146* or *23965*
- Transaction number keyword: *highest*

When using the **-t** option, take into account the possibility of a timewarp. The `AC_SYNC` environment variable (see [AccuRev User Preferences](#) on page 6) determines how a timewarp will be handled. See also section [System Clock Synchronization](#) on page 31 of the *AccuRev Administrator's Guide*.

Specifying the Transactions By default, **hist** displays the entire transaction history of the element(s), or of the entire depot. The various forms of the **-t** option restrict the display to a single transaction, or to a range of transactions. Each variant involves one or two time-specs. A single time-spec means "the transaction that took place at this time" or "the most recent transaction that took place before this time".

hist

Two time-specs define an interval; *hist* reports all the transactions in the interval that involve the element(s) you've specified.

Examples

Display the transaction history for each element in the current directory:

```
> accurev hist *
```

Display the transactions involving the current directory itself:

```
> accurev hist .
```

Display transactions 145 through 176, inclusive:

```
> accurev hist -t "176-145"
```

Display the five most recent transactions, with a verbose listing of *keep* and *create* transactions:

```
> accurev hist -t now.5 -fv
```

Display all the transactions for element **base.cc** that involved stream **gizmo_test**:

```
> accurev hist -s gizmo_test base.cc
```

Display all the transactions for element **base.cc** that were executed by users *other than* john smith and jones:

```
> accurev hist -u "john smith,jones" --exclude_users base.cc
```

Display information on the most recent *promote* transaction, including the *keep* transactions that preceded it:

```
> accurev hist -t now -k promote -fe
transaction 113; promote; 2007/02/19 17:55:55 ; user: john
  \.\src\brick.h 13/2 (16/2)

  # add RIVER
  version 16/2 (16/2)
  ancestor: (16/1)

  # remove ALLOCSIZE
  version 16/1 (16/1)
  ancestor: (15/1)
```

Note that the output of this command can be in XML format if you specify **-fex** instead of **-fe** at the command line. The **<streams>** element details all the dynamic streams and workspace streams involved in the reported transaction(s). A transaction's timestamp (**time** attribute of the XML element **<transaction>**) is reported as the number of seconds since Jan. 1, 1970 UTC. *Special Field Types* on page 276 describes a technique for converting this timestamp into a human-readable string.

*Important! If you have been parsing the XML output from the **hist** command, you will need to check that you are getting the correct **<stream>** tag, since that tag now appears within both the **<transaction>** and the **<streams>** tags. The content of the new **<stream>** tag will vary depending on the AccuRev command recorded in the transaction.*

hist

Display issues related to a version of an element at the time of promote transactions, and format the output in XML. Note the `<historic_only>` tag at the end, indicating that the version is no longer the head, but was at the time of the transaction.

```
> accurev hist -fix -a -k promote
```

```
<AcResponse
  Command="hist"
  TaskId="1032">

....

<transaction
  id="11"
  type="promote"
  time=""
  user="testuser1">
  <version
    path=".\file_0003"
    eid="4"
    virtual="2/2"
    real="4/2"
    virtualNamedVersion="s22245/2"
    realNamedVersion="ws22245_1_testuser1/2"
    elem_type="text"
    dir="no">
    <issueNum>2</issueNum>
  </version>
</transaction>
<transaction
  id="9"
  type="promote"
  time=""
  user="testuser1">
  <version
    path=".\file_0003"
    eid="4"
    virtual="2/1"
    real="4/1"
    virtualNamedVersion="s22245/1"
    realNamedVersion="ws22245_1_testuser1/1"
    elem_type="text"
    dir="no">
    <issueNum
      historic_only="true">2</issueNum>
  </version>
</transaction>
```

hist

Display all AccuWork ("dispatch") transactions in XML format, using verbose mode. Note that any *cpk add* and *cpk remove* transactions are flagged in the XML output with *cpk=* attributes in the `<transaction>` tag.

```
> accurev hist -fvx -a -k dispatch

<AcResponse
  Command="hist"
  TaskId="1036">
  <transaction
    id="16"
    type="dispatch"
    time=""
    user="testuser1"
    cpk="add">
    <issues>
      <issue>
        <issueNum>3</issueNum>
        <elements>
          <element
            eid="2"
            member="1"
            head_version="4/2"
            basis_version="0/0"
            name="/file_0001"/>
          </elements>
        </issue>
      </issues>
      <streams>
        <stream
          id="4"
          name="ws22245_1_testuser1"
          type="workspace"/>
        </streams>
      </transaction>
    <transaction
      id="15"
      type="dispatch"
      time=""
      user="testuser1"
      cpk="remove">
      <issues>
        <issue>
          <issueNum>3</issueNum>
          <elements>
            <element
              eid="2"
```

hist

```
        member="0"  
        name="/file_0001"/>  
    </elements>  
  </issue>  
</issues>  
</transaction>
```

Display in the xml output the state of the change package related to transaction #2123 as it was at the time of the transaction:

```
> accurev hist --show_historic_cpk -t 2123 -fx
```

See Also

translist

Techniques for Selecting Elements on page 10

XML Output Reference on page 259

hostinfo

hostinfo

display license server information

Usage

```
accurev hostinfo
```

Description

The *hostinfo* command displays the following information about the AccuRev license server:

- **Host name** -- The name of the machine hosting the AccuRev license manager.

Options

There are no options for the *hostinfo* command.

Examples

Display information about the AccuRev license server:

```
> accurev hostinfo
```

incl set rule to include elements in a workspace or stream

Usage

```
accurev incl [ -s <stream> ] [ -b <backing-stream> ] [ -fx ]
           { -l <list-file> | <element> }
```

Description

The *incl* command includes the specified element (either a directory or a file) in a particular workspace or stream. If you don't use the *-s* option to name a stream, *incl* operates on the workspace containing the current working directory.

You may need to use a complete depot-relative pathname — starting with *./* (UNIX/Linux) or *\.* (Windows) — to specify the element to be included. A path must exist in the workspace or stream from the depot's root directory to the element you wish to include. That is, all the directories between the root directory and your element must be included in the workspace or stream. One or more *incldo* commands might be needed to satisfy this requirement.

Note: As of Version 4.6, an *incl* command automatically causes AccuRev to execute *incldo* commands, as necessary, to include intermediate directory levels in the workspace or stream.

Including a directory element causes it to appear in the workspace or stream, along with all the files and subdirectories below the directory element. Including a file element causes just that one file to appear.

Inheritance of Inclusion

When you include elements in a workspace, the appropriate versions of the files are immediately copied to the workspace tree.

When you include elements in a higher-level stream, the inclusion instantly applies to streams below it in the depot's stream hierarchy. The inclusion also applies to workspaces below the stream, but the elements don't appear in a workspace until you *update* it. This inheritance of inclusions down the stream hierarchy stops at snapshots and at dynamic streams whose basis time precedes the time of the inclusion. The element inclusions don't apply to the snapshot (which is, by definition, immutable) or to that time-based stream.

Creating an Empty Workspace Before using 'incl'

Using *mkws* with the *-i* option (to create an initially empty workspace) is a useful technique for organizations that use *incl* and *excl* to control which elements appear in a workspace.

Cross-Links: Including Elements from a Stream other than the Backing Stream

The *-b* option enables you to change the backing stream for individual elements. The element is included in your workspace (or in the dynamic stream you specify with *-s*); when a new version of the element arrives — through an *update* for a workspace, or through automatic inheritance for a dynamic stream — the version comes from the *-b* stream (or snapshot), not from the original backing stream.

The *incl* command is said to create a cross-link from your workspace (or the *-s* stream) to the *-b* stream. The *files* and *stat* commands list a cross-linked element as having (**xlinked**) status. When you cross-link a directory element, the entire subtree of elements below it also become cross-linked.

incl

Cross-linking provides read-only access to elements. You can view or execute the contents of the version that is brought into your workspace or stream. But you can't use commands like *keep* or *promote* on the element.

As of AccuRev 4.5: in a stream, you cannot *promote* cross-linked elements; in a workspace, you cannot *keep* (or *anchor*, or *defunct*, etc.) cross-linked elements.

Notes: for cross-linking to succeed, the element must have been present in the stream specified with *-b* at the time your workspace was last updated. In a time-based stream, you cannot create a cross-link to another time-based stream (or to a snapshot) whose basis time is “in the future” with respect to your stream.

Options

-s <stream>

Apply the include rule to the specified stream. When using this option, you cannot specify the element to be included with a simple filename; use its depot-relative pathname instead.

-b <backing-stream>

For this element, create a cross-link (xlink) from the current workspace (or from the *-s* stream) to the specified dynamic stream or snapshot. The element must already appear both in your workspace and in the “target” stream or snapshot.

-fx Display the results in XML format.

-l <list-file>

Process the elements listed in <list-file>. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in a list-file.

If you use this option, any <element> specified is silently ignored.

<element>

An element name. If you also specify a <list-file> using the *-l* option, this <element> is silently ignored.

Examples

Include the **python** subdirectory of the previously excluded **tools** directory in the current workspace:

```
> accurev incl \.\tools\python
```

Include the **src\mmgt** subdirectory in the current workspace, getting its versions from the snapshot **widget_V3.4**:

```
> accurev incl -b widget_V3.4 \.\src\mmgt
```

See Also

clear, *excl*, *includo*, *lsrules*, *update*

Techniques for Selecting Elements on page 10

incldo

set rule to include a directory, not its contents, in a workspace stream

Usage

```
accurev incldo [ -s <stream> ] [ -fx ] { <directory-element> |
  -l <list-file> }
```

Description

The *incldo* command supports the ability of AccuRev to include files and directories that are deep in a depot's directory hierarchy, without having to include lots of other, unwanted elements. For example, suppose a depot contains a **tools** subdirectory, which contains a **scripts** subdirectory, which contains a **unix** subdirectory, which contains two subdirectories, **perl** and **python**. If you need only the **python** subdirectory, not the rest of the **tools** subtree, you can enter these commands:

```
accurev incldo tools
accurev incldo tools/scripts
accurev incldo tools/scripts/unix
accurev incl tools/scripts/unix/python
```

Note: As of Version 4.6, you need not enter the *incldo* commands for the **scripts** and **unix** directories — AccuRev automatically executes the required *incldo* commands to “traverse” intermediate directory levels.

Note that *incldo* includes a directory, but merely for the purpose of providing a path to some data lower in the directory hierarchy. *incldo* actually serves to *exclude* the entire contents of the specified directory; subsequent *incl* commands can override this exclusion for particular files/directories under it.

To clarify the preceding paragraph, here's an example. Suppose directory **widget** contains subdirectories **devel**, **test**, and **support**. The command “incldo widget”:

- Includes the directory **widget** in your workspace (or in the stream you specify with *-s*). The directory is empty.
- Enables the subsequent execution of *incl* or *incldo* commands on the subdirectories: “incl devel” or “incldo devel”, “incl test” or “incldo test”, “incl support” or “incldo support”.

Keep in mind the relationship between the update level of the workspace you're modifying with *incldo* (or the basis time of the stream you're modifying). For example, a subsequent “incl devel” command will succeed only if subdirectory **devel** existed in directory **widget** at the time of the workspace's most recent update.

Options

-s <stream>

Apply the include-directory-only rule to the specified stream. If you use this option, you must specify the directory element using a depot-relative pathname.

-fx Display the results in XML format.

-l <list-file>

Process the elements listed in *<list-file>*. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing

incldo

white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in a list-file.

If you use this option, any *<directory-element>* specified is silently ignored.

<directory-element>

A directory element. If you also specify a *<list-file>* using the **-l** option, this *<directory-element>* is silently ignored.

Examples

Include just the top-level **readme.txt** file and the **online** subdirectory from the hierarchy **widget\support\documentation\online**:

```
> accurev incldo widget
> accurev incl widget\support\documentation\online\readme.txt
```

See Also

clear, excl, incl, lsrules, update

Techniques for Selecting Elements on page 10

info

info show basic information about the current session

Usage

```
accurev info [ -v ]
```

Description

At its most basic level, the *info* command displays the values for the following characteristics of the AccuRev user environment:

```
Principal
Host
Server name
Port
DB Encoding
ACCUREV_BIN
Client time
Server time
```

When executed from within a workspace directory, *info* also displays the following additional fields:

```
Depot
Workspace/ref
Basis
Top
```

When executed with the -v (verbose) option, *info* also displays the following additional fields:

```
client_ver
server_ver
```

Options

- v Display additional information: the protocol version numbers of the AccuRev client and server programs. **Note:** This option does not display major/minor version numbers. To check the version of your client, simply type `accurev` with no arguments. To check the version of the server, examine the top dozen or so lines of a server log file such as `<AccuRev-installation-dir>/storage/site_slice/logs/acserver-<timestamp>.log`

Examples

Display all possible info fields, by using -v from within a workspace directory:

```
> cd \workspaces\domestic_na
> accurev info -v

Principal:      bfrederick
Host:          chungas
client_ver:    126
Server name:   chungas
```

info

```
Port:          5050
DB Encoding:   Unicode
ACCUREV_BIN:   C:/Program Files/ac_5_2c/bin
server_ver:    126
Client time:   2011/06/09 15:03:11 Eastern Daylight Time (1307646191)
Server time:   2011/06/09 15:03:11 Eastern Daylight Time (1307646191)
Depot:         domestic_na
Workspace/ref: domestic_na_bfrederick
Basis:         domestic_na
Top:           C:/workspaces/domestic_na
```

See Also

start

ismember

ismember check if named user is a group member

Usage

```
accurev ismember <principal-name> <group-name>
```

Description

The *ismember* command determines whether an AccuRev user (specified by principal-name) belongs to a specified AccuRev group. It displays **1** if the user is a member of the group, and **0** if not.

Note: the command's return value is 0 in either case.

Examples

Is AccuRev user **john** a member of AccuRev group **ctdvt**?

```
> accurev ismember john ctdvt
1
```

See Also

addmember, *rmmember*, *show*

issuediff compare two streams in terms of their change packages

Usage

```
accurev issuediff -s <from-stream> -S <to-stream> [ -fx ]
```

Description

The *issuediff* command compares the change packages in two streams, snapshots, or workspaces (for simplicity, we'll just say "stream"), showing which changes that are included in the "from" stream have not yet been propagated to the "to" stream. The output is a list of issue numbers for those change packages.

Options

-s *<from-stream>* **-S** *<to-stream>*

The streams, snapshots, or workspaces to be processed.

-fx Display the results in XML format.

Examples

List the issues whose change packages are completely in stream **brass_dvt**, whose changes have not yet been propagated to stream **brass_tst**. Display the output in XML format.

```
> accurev issuediff -s brass_dvt -S brass_tst -fx
<acResponse>
  <issues>
    <issue>
      <issueNum
        fid="1">3</issueNum>
      <transNum
        fid="2">272</transNum>
      <targetRelease
        fid="21">rel2.0</targetRelease>
    ...
```

See Also

cpkadd, *cpkdepend*, *cpkdescribe*, *cpkremove*, *issuelist*, *patch*

issuelist list the change packages in a stream

Usage

```
accurev issuelist [ -s <stream> | -p <depot> ] [ -a ] [ -i ]
[ -I <issue-number> [ -d ] [ -3 ] ] [ -t <time-spec> ] [ -fx ]
```

Description

The *issuelist* command displays a list of issue numbers: for the issues whose change packages are partially or completely “in” a particular stream, snapshot, or workspace. (For simplicity, we just say “stream” below.)

- Each change package entry is said to be “in” a particular stream if its head version (Version column) is the same as — or is an ancestor of — the version in the stream.
- A change package is said to be “completely in” a particular stream if all of its entries are “in”.
- A change package is said to be “partially in” a particular stream if some — but not all — of its entries are “in”.
- A change package is said to be “active” in a particular stream if at least one of its head versions is in the stream’s default group.

Options

- a** Include an issue even if none of its change package’s head versions is active in the stream. By default, issues with such “old” change packages are excluded from the listing.
- d** Display information about the issue specified by the *-I* option, even if it is not in a change package for the specified stream or any stream in the specified depot.
- i** (“incomplete”) List issue records whose change packages are “partially in” the stream. By default, issue records whose change packages are “completely in” the stream are listed.
- I <issue-number>**
Restrict the listing to the specified issue record. No error occurs if that issue record’s change package would not be listed; the command output is empty.
- 3**
Specifies that the <issue-number> specified by the *-I* switch is a third-party ITS key rather than an AccuWork issue number.
- p <depot-name>**
The depot to be processed. Returns all the issue record numbers for all the change packages for all the streams in the depot.
- s <stream>**
The stream, snapshot, or workspace to be processed. Default: the workspace containing the current working directory.

issuelist

-t <*time-spec*>

Display the issues that existed as of the specified time. A *time-spec* can be any of the following:

- Time in <"YYYY/MM/DD HH:MM:SS"> format, "**2007/08/07 20:27:15**", for example. Note that you must use quotes when specifying time-spec as *YYYY/MM/DD HH:MM:SS* to ensure that the entire argument is interpreted by the command shell as a single token.
- Time keyword: **now**
- Transaction number as a positive integer: e.g. **146** or **23965**
- Transaction number keyword: **highest**

-fx Display the results in XML format. The results include the field-by-field contents of each of issue record, not just the issue number.

Examples

List the issues whose change packages are completely in workspace **brass_dvt_mary**, including those with no version active in that workspace.

```
> accurev issuelist -s brass_dvt_mary -a  
Issue: 1  
Issue: 4  
Issue: 5  
Issue: 17
```

See Also

[cpkadd](#), [cpkdepend](#), [cpkdescribe](#), [cpkremove](#), [issuediff](#), [patch](#)

keep create a new version of an element

Usage

```
accurev keep [ -c <comment> ] [ -dmn ] [ -E <value(s)> ] [ -O ]
[ -R ] { -l <list-file> | <element-list> }
```

Description

The **keep** command creates a new version of an element, using the copy of the element that is currently in your workspace. It is similar to the “check-in” command of other configuration management systems.

In order to **keep** an element, it must exist as an element in AccuRev and your workspace must have a copy of the element in it. If the element is not yet in the repository (e.g. you just created it with a text editor), you must first **add** it. **add** performs a **keep** automatically, creating version 1 in your workspace stream.

You usually do not need to perform any special command, such as a “check-out”, before or after performing a **keep**. After you have kept an element, you can immediately edit it again.

Keeping a version transitions an element from the modified state to the kept state. The element won’t be selected by the **-m** option of any of **accurev** command until you modify the copy in your workspace again.

Exclusive File Locking and Anchor-Required Workspaces

See [File Locking in Workspaces](#) on page 5.

Pre-Operation Triggers

If a **pre-keep-trig** trigger is defined for the depot, it fires before the **keep** command is executed. Similarly, an **admin_preop_trig** script runs on the AccuRev Server machine before the **keep** command is executed. The server-side script can (for example) control the setting of the element(s)’ exclusive file locking state, overriding the **keep -E** specification. See the [mktrig](#) reference page and [AccuRev Triggers](#) on page 77 of the *AccuRev Administrator’s Guide*.

‘Keep’ and Directories

AccuRev creates a new version of a directory element when you rename the directory or move it to another location in the depot’s directory hierarchy (**move** command). You never need to explicitly **keep** a directory.

Options

-c <comment>

Specify a comment for the transaction. The next command-line argument should be a quoted string. Alternatively, the next argument can be in the form **@<comment-file>**, which uses the contents of text-file **<comment-file>** as the comment.

Default: enter a comment interactively, using the text editor named in environment variable EDITOR (or a system-dependent default editor).

-d Select all elements in your default group.

-E <value(s)>

Change the element type to **text**, **ptext**, or **binary**. This change applies to the version being created and establishes the default type for future versions; it does not change the type of any existing version of the element.

keep

You can also use this option to change the exclusive file locking state for this element: *serial* or *parallel*.

To specify two (non-conflicting) values at once, separate them with a comma, but not a SPACE:

```
-E serial,binary
```

See [Controlling the Element Type and Exclusive File Locking State](#) in the [add](#) reference page.

-R Recurse into the directories specified in `<element-list>`, and **keep** all modified files in the directory that are not in the default group. (You must also specify **-n**, and `<element-list>` must consist of a single directory name. Use “.” to specify the current working directory.)

-l `<list-file>`

Process the elements listed in `<list-file>`. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any `<element-list>` specified is silently ignored.

`<element-list>`

One or more element names, separated by whitespace. If you also specify a `<list-file>` using the **-l** option, this `<element-list>` is silently ignored.

-m Select only the modified elements.

-n Select only the modified elements that are *not* in your default group.

-O Override any errors on modification-time timewarps. Allows a **keep** on a file with a modification time older than its ancestor's modification time.

Examples

Create a new version of a particular element, specifying a comment string:

```
> accurev keep -c "fixed typos in greeting message" greet.msg
```

Create new versions of all modified elements that are not in the workspace's default group:

```
> accurev keep -n
```

Create new versions of all modified elements that are in the workspace's default group:

```
> accurev keep -m -d
```

Create new versions of all modified elements, regardless of whether or not they are in the default group:

```
> accurev keep -m
```

See Also

[add](#), [ln](#), [mktrig](#), [move](#)

[Techniques for Selecting Elements](#) on page 10

licenses

display or set AccuRev licensing information

Usage

```
accurev licenses [ -fx[v] ] [ -u ]
```

Description

The *licenses* command displays the AccuRev licensing information currently in effect, or updates the licensing information by rereading the license file.

Options

-u Update the licensing information by rereading file **aclicense.txt** in the directory *<AccuRev-installation-dir>/storage/site_slice* on the AccuRev Server machine. Then display the updated results.

-fx Display the results in XML format.

-fxv (“verbose”) For floating, 7-day, and 24-hour licenses, include the following information:

- Usage count
- Minimum checkout time (in minutes)
- Information about the users who currently have licenses: name, when the license was originally checked out, and when it was last renewed. Times are reported in the server’s time zone.

Note: If the time of last renewal is longer ago than the minimum checkout time, that could indicate that a non-expiring session is active for the user.

Examples

Display the current AccuRev licensing information, in XML format (fields in italics constitute the verbose output):

```
> accurev licenses -fxv
<acResponse>
  <licenseCounts>
    <license
      name="AccuRev"
      count="250"
      usage_count="1"
      min_checkout="1440">
      <user
        name="jclark"
        checkout_time="2016/10/05 16:10:14"
        last_renewed_time="2016/10/05 16:10:14"/>
    </license>
```

licenses

```
</licenseCounts>  
</acResponse>
```

See Also

AccuRev Installation and Release Notes

ln

ln create or change an element link or symbolic link

Usage

```
accurev ln [ -s ] [ -c <comment> ] [ -p ] <target> <link-name>
accurev ln -i <target>
```

Description

The **ln** command creates a new element link or symbolic link element, or changes the target of an existing link element. The link element's status becomes **(kept)(member)**. As with other kinds of element, you must *promote* a newly created or changed link in order to make it visible to other developers.

Although it can create a new element or change an existing one, the **ln** command does not cause a *pre-create-trig* or *pre-keep-trig* trigger to fire.

With the **-i** option, the **ln** command lists all the link elements in the workspace that point, directly or indirectly, to the specified target element.

The **add** command can also create element link and symbolic link elements, by converting existing link objects.

Element Links

An element link (or, more precisely, an “element-link element”) is an element whose content is a reference to another element. An element link is instantiated in a workspace tree or reference tree as a UNIX/Linux or Windows (on NTFS only) hard link.

Symbolic Links

A symbolic link (or, more precisely, a “symbolic-link element”) is an element whose content is a pathname. AccuRev does not check the validity of this pathname when the link is created or changed. Thus, any of the following might be true:

- The target is an existing element (file, directory, or link) in the workspace.
- The target is an **(external)** object in the workspace.
- The target pathname is within the workspace, but no object currently exists at the pathname.
- The target is not within the workspace at all (for example, **c:\temp\junk** or **/tmp/foo**). An object may or may not currently exist at the pathname.

On UNIX/Linux, a symbolic link element is instantiated in a workspace tree or reference tree as a UNIX/Linux symbolic link.

On Windows, a symbolic link element for directories only (and on NTFS file systems only) is instantiated as a Windows junction point.

Important! On Windows, an attempt to create a link to a non-directory element on NTFS, or to any element on a non-NTFS (FAT, FAT32, NFS, etc.) file system will create the symbolic-link element but not the symbolic link itself, and display a warning similar to the one shown below.

```
Warning: The symbolic-link element was added/modified successfully, but the
symbolic-link could not be instantiated on the local file-system:
```

```
\\.mylink->\\.dir05\sub01
```

When updating a workspace located in a non-supported file system, an existing link to a directory generates an error (and **update** creates an empty, standard directory at the target location):

```
Re-linking "dir01sub03.mylink" - failed
```

Processing of Links by Commands

In general, an operating system command that processes a file's contents (such as Windows **type** or **copy**, or UNIX/Linux **cat** or **cp**) will traverse an element link or symbolic link to a file. That is, if you specify the link object to the command, it will process the target of the link. Similarly, the Windows command **dir** and the UNIX/Linux command **ls** will traverse an element link or symbolic link to a directory.

In general, an AccuRev command, such as **promote**, processes a link object itself, not its target.

Options

-c <comment>

Specify a comment for the transaction. The next command-line argument should be a quoted string. Alternatively, the next argument can be in the form **@<comment-file>**, which uses the contents of text-file **<comment-file>** as the comment.

- i** List all element-link elements in the workspace that point to the specified target element, either directly or indirectly. Each such link element is reported on a separate line, in this format:

```
link-name -> direct-target-name
```

Both *link-name* and *direct-target-name* are reported as depot-relative pathnames. For a chain of links: **L -> I1 -> I2 -> ... -> T**, each intermediate link (I1, I2, etc.) will be the *direct-target-name* on one line and the *link-name* on another line.

- p** (for use with **-s** when the target is a depot-relative pathname) Converts the target pathname, which begins with **./** or **\.**, to a relative pathname.

UNIX/Linux: do not use **-p** if the target pathname begins with **./**, but is actually an absolute pathname.

- s** Create a symbolic link. Default: attempt to create an element link. Returns an error if the target is not in the workspace.

<target>

When you're creating an element link (no **-s** option), the target must be an element in the same workspace.

When you're creating a symbolic link (**-s** option), you can use any kind of pathname to specify the target: simple filename, relative pathname, depot-relative pathname, or absolute. The contents of the symbolic link element is the pathname, exactly as you entered it (with one exception — see below).

ln

More notes on specifying the target of a symbolic link:

- You might need to enclose a relative pathname in quotes, to prevent the operating system from expanding the pathname to an absolute (full) pathname.
- If the target does not currently exist, the newly created symbolic link element will have **(missing-target)** status.
- Only an absolute pathname can indicate a location outside your workspace.
- If you specify the target as a depot-relative pathname, you must also use the **-p** option; the target is automatically converted to a relative pathname, to enable operating-system commands to traverse the link:

```
$ pwd
/user/wks/john/dir02/sub03
$ accurev ln -p -s ../dir00/sub00/file00.txt mylink
Created symbolic-link ../dir02/sub03/mylink->../../dir00/sub00/file00.txt
```

<link-name>

The element link or symbolic link object to be created. If <link-name> already exists, its target is changed to <target>. You cannot create a circular chain of element links. You cannot overwrite an element link with a symbolic link, or vice-versa.

Note: Running "`accurev ln directoryName linkName`" will create a UNIX/Linux symbolic link, even though AccuRev creates an **(elink)**.

Examples

Create an element link named **relnotes**, pointing to file **release_notes.txt**, which resides two directory levels up:

```
> accurev ln "../../release_notes.txt" relnotes
```

List all element links that point, directly or indirectly, to file **release_notes.txt**:

```
> accurev ln -i \.\doc\release_notes.txt
\.\install\README.txt->\.\doc\release_notes.txt
\.\README.txt->\.\install\README.txt
```

See Also

add, incl (to create cross-links between streams)

Techniques for Selecting Elements on page 10

lock lock a dynamic stream against promotions

Usage

```
accurev lock [ -c <comment> ] [ -kf | -kt ]
           [ { -e | -o } <principal-name-or-group-name> ] <stream>
```

Description

The **lock** command prevents users from making various changes to the specified dynamic stream:

- With no **-k** option, the command creates an “all” lock, which:
 - prevents users from promoting versions either *to* or *from* the specified stream
 - prevents users from modifying the contents of a stream with an **incl**, **excl**, or **incldo** command
 - prevents users from modifying the specifications of a stream with a **chstream** command
- With **-kf**, the command creates a “from” lock, which prevents users from removing versions from the stream’s default group. This means that users cannot:
 - promote versions *from* the specified stream to other streams
 - purge versions from the stream
- With **-kt**, the command creates a “to” lock, which prevents users from adding versions to the stream’s default group, for example, via promoting versions *to* the specified stream from other streams.

Usage of both **-kf** and **-kt** in the same command is not valid (the first option appearing on the command line is taken, and the second is ignored). But you can use these two options in separate **lock** commands, to establish both “from” and “to” locks on the same stream.

By default, the lock applies to all users. The **-o** *<principal-name>* option makes the lock apply only to a specified AccuRev user. Similarly, the **-o** *<group-name>* option makes the lock apply only to the members of the specified AccuRev group.

Conversely, use the **-e** option to make the lock apply to everyone but the specified AccuRev user or group. That is, the specified user or group will be able to perform promotions, but no one else will.

Limitations on Locking

A stream can have at most one lock of each type — “all”, “from”, or “to”. This means you cannot use multiple **lock -o** commands to lock a stream for users **tom**, **dick**, and **harry**. To accomplish this, put these users in an AccuRev group and apply a lock to that group.

For a given user, an “all” lock overrides “from” and/or “to” locks.

An “all” lock on a stream does not prevent the stream from being deactivated with **remove**.

Locks and Virtual Versions

AccuRev’s system of real versions and virtual versions means that a version of an element seems to exist in multiple streams at once. A lock on one of those streams does not apply to any of the other streams. For example, suppose you create a real version of a file in workspace stream **gizmo_dvt_derek**, and then promote this version to stream **gizmo_dvt**. If you put a “from” lock on stream **gizmo_dvt_derek**, you can still promote the version from stream **gizmo_dvt** to another stream.

lock

Removing Locks

Use the *unlock* command to remove an existing lock. Placing a new lock on a stream automatically removes any existing lock of the same type. For example, if you issue the following commands, the result will be a “to” lock for everyone and a “from” lock for everyone except **john**.

```
accurev lock -kf -e tom stream1
accurev lock -kt stream1
accurev lock -kf -e john stream1
```

Options

-c <comment>

Specify a comment that will be displayed in the future when this lock prevents a user from promoting a version. The next command-line argument should be a quoted string. Alternatively, the next argument can be in the form *@<comment-file>*, which uses the contents of text-file *<comment-file>* as the comment. Default: no comment is associated with this lock.

-kf (“from” lock) Disallow promotions *from* the stream and purging versions in the stream. Cannot be used with *-kt*.

-kt (“to” lock) Disallow promotions *to* the stream. Cannot be used with *-kf*.

-o *principal-name-or-group-name*

Make the lock apply only to the specified AccuRev user, or only to the members of the specified AccuRev group. Cannot be combined with *-e*.

-e *principal-name-or-group-name*

Make the lock apply to everyone but the specified AccuRev user, or to everyone but the members of the specified AccuRev group. The specified user or group will be able to perform promotions, but no one else will. Cannot be combined with *-o*.

Examples

Disable all promotions to and from stream **tulip_dvt**; also disable include/exclude changes and *chstream* changes:

```
> accurev lock tulip_dvt
```

Disable all promotions from stream **tulip_dvt**:

```
> accurev lock -kf tulip_dvt
```

Allow user **mary** to make promotions to stream **tulip_dvt**, but disallow everyone else from doing so:

```
> accurev lock -kt -e mary tulip_dvt
```

See Also

chstream, *promote*, *remove*, *unlock*

login

log in to an AccuRev Server

Usage

```
accurev login [ -n ] [ <user-name> [ <password> ] ]
```

Description

The *login* command establishes your AccuRev user identity. You must specify the name of a registered AccuRev user, either on the command line, or when the *login* command prompts you. Similarly, you can enter a password (case-sensitive) on the command line or when prompted. To indicate “no password” on the command line, enter a null string — for example, with two consecutive double-quote characters.

You won’t be able to execute most AccuRev CLI commands unless you are logged in. The exceptions are *help*, *login*, *info*, and *secinfo*. In addition, exactly one *mkuser* command is allowed in a new AccuRev installation, to prevent a chicken-and-egg authentication problem.

By default, a successful login initiates a session that lasts four hours. When the session expires, you’ll have to log in again. The *-n* option creates a non-expiring session. (For security reasons, we suggest using the option sparingly.)

The Session File

A successful *login* command creates an encrypted file that records your AccuRev username and password, along with the IP address of your client machine. Most AccuRev client commands can be executed only by an authorized user. Such commands send the information in your session file to the AccuRev Server process, so that you don’t need to repeatedly “remind” the AccuRev Server who (and where) you are.

If you’re already logged in, and you successfully log in — as the same user or a different user — the existing session file is overwritten.

Session Expiration

Session files are automatically deleted by the AccuRev Server after an administrator-configurable interval (default: 240 minutes)

Multiple Session Files for Different Servers

The name of the session file includes the hostname and port number of the AccuRev Server. If you *login* to different AccuRev Server processes, you’ll have multiple session files, one for each Server. Example:

```
session_VENUS_5050
session_MARS_5050
session_MARS_5999
```

These session files indicate that your are logged in to an AccuRev Server on host **venus**, listening on port **5050**, and are also logged into two different AccuRev Server processes on host **mars**.

If you intend to maintain connections to two or more AccuRev Server processes, you’ll probably want to use the *-H* option in your *accurev* commands. See [Working with Multiple Repositories](#) on page 4.

Multiple Session Files for the Same Server

What if you want to be testuser **john** in one command shell, but testuser **mary** in another — both using the same AccuRev Server? You can’t have two session files with the same name (for example, **session_VENUS_5050**) in your **.accurev** subdirectory. But you can have two or more session files with the same name in different **.accurev** subdirectories.

login

The **login** command uses the value of environment variable ACCUREV_HOME as the pathname of the parent of the **.accurev** subdirectory (and will create this subdirectory, if necessary). So do the commands that authenticate you by checking for the existence of a session file.

Options

- n** Create a login session that does not expire. Logins created with this option will persist, even through a restart of the AccuRev Server.

Examples

Log in, letting the command prompt for both your username and password. For security, your keystrokes are echoed as “*” characters.

```
> accurev login
Username: john
Password: *****
```

Log in, specifying your username, but letting the command prompt for the password:

```
> accurev login john
Password: *****
```

Log in, specifying your username and password on the command line:

```
> accurev login john Every1CanSeeThis
```

Log in, specifying your username and indicating that you don't have a password:

```
> accurev login john ""
```

See Also

[authmethod](#), [info](#), [logout](#), [mkuser](#), [secinfo](#)

[User Authentication](#) on page 57 of the *AccuRev Administrator's Guide*

logout

logout

log out from an AccuRev Server

Usage

```
accurev logout
```

Description

The *logout* command cancels a login session, removing the session file from your **.accurev** subdirectory. Sessions are also canceled when you exit the AccuRev GUI, if the **Keep Session Active** preference is enabled. Administrator's can also cancel all active sessions for a user using the *remove* command.

See Also

login, remove

lsacl show access control list entries

Usage

```
accurev lsacl [ -fx ] { depot | stream } [ <depot-name> | <stream-name> ]
```

Description

The *lsacl* command lists access control list entries (“ACL permissions”) that pertain to depots, or entries that pertain to streams. (For ACLs that pertain to individual elements, see the *eacl* command.) The keyword **depot** or **stream** is required. You can optionally restrict the listing to a particular depot or stream by specifying its name as the final command-line argument.

In the Group column of the listing:

- **authuser (b)** indicates all users who have passwords.
- **anyuser (b)** indicates all users who do not have passwords.

The **(b)** annotation makes it easy to distinguish these “built-in” user categories from individual named users.

The Inheritable column of the listing indicates whether or not the permission was created with the *setacl* command’s **noinherit** keyword (default) or the **inherit** keyword.

Options

-fx Display the results in XML format.

```
<AcResponse
  Command="lsacl"
  TaskId="66032">
  <Element
    Name="acl6"
    Group="development users"
    Type="group"
    Rights="all"
    Inheritable="true">
  </AcResponse>
```

Examples

List the ACL permissions that pertain to all depots:

```
> accurev lsacl depot
```

List the ACL permissions that pertain to the stream **widget_tst**:

```
> accurev lsacl stream widget_tst
```

See Also

addmember, *chpasswd*, *eacl*, *mkgroup*, *mkuser*, *setacl*

lsrules

show the include/exclude rules for a workspace or stream

Usage

```
accurev lsrules [ -s <stream> ] [ -d | -R ] [ -t <time-spec> ]
[ -f <format(s)> ]
```

Description

The *lsrules* command displays the include/exclude rules for your workspace, or for a stream that you specify with the *-s* option. By default, the listing also includes the rules that apply to the workspace (or stream) by virtue of being inherited from higher-level streams. Such rules are listed with *-s <stream>* to indicate the stream in which they were set.

Note: Setting an include/exclude rule for a workspace affects it immediately. Accordingly, *lsrules* lists the rules that are currently in effect for a workspace, not the rules that were in effect at the time of the most recent *update*.

To restrict the listing to the rules that were explicitly set in your workspace (or stream), use the *-d* option. Conversely, use *-R* in combination with *-s* to list the rules set in all streams and workspaces in an entire stream subhierarchy.

Options

-s <stream>

Display the include/exclude rules for the specified stream.

-d Display only the rules that were explicitly set for the workspace (or stream). Don't display rules that apply because they're inherited from higher-level streams.

-R Display the rules set for each stream and workspace in the subhierarchy below the stream specified with *-s*.

Note: *-R* may *not* be used in conjunction with the *-fmx* option.

-t <time-spec>

Display the rules set that existed as of the specified time. A *time-spec* can be any of the following:

- Time in *<"YYYY/MM/DD HH:MM:SS">* format, "**2007/08/07 20:27:15**", for example. Note that you must use quotes when specifying time-spec as *YYYY/MM/DD HH:MM:SS* to ensure that the entire argument is interpreted by the command shell as a single token.
- Time keyword: **now**
- Transaction number as a positive integer: e.g. *146* or *23965*
- Transaction number keyword: *highest*

-f <format(s)>

Use one or more of the following format letters:

c: Restrict the listing to cross-link (*incl -b*) rules.

x: Display the results in XML format.

lsrules

m: (must specify **x**, also) Make the XML-format output suitable for input to the **mkrules** command. This makes it easy to copy a set of include/exclude rules from one stream to another. See the [mkrules](#) reference page for instructions on how to edit this output.

Note: The **-fmx** option may be used with the **-d** option, but *not* with the **-R** option.

Examples

List the include/exclude rules for the current workspace, excluding rules inherited from higher-level streams:

```
> accurev lsrules -d
```

List, in XML format, the include/exclude rules for stream **gizmo_int**, along with rules inherited from higher-level streams:

```
> accurev lsrules -s gizmo_int -fx
```

See Also

[excl](#), [incl](#), [mkrules](#), [show](#) (streams and wspaces)

[XML Output Reference](#) on page 259

merge merge changes from another stream into the current version of an element

Usage

```
accurev merge [ -v <ver-spec> ] [ -G ] [ -K ] [ -O ][--
use_time_basis_for_overlap]{ [ -Fx ] -l <list-file> | <element-list>}

accurev merge -o [ -G ] [ -K ] [ -O ][--use_time_basis_for_overlap]

accurev merge -i [ -B ] [ -v <ver-spec> | -s <stream> ][--
use_time_basis_for_overlap]{ [ -Fx ] -l <list-file> | <element-list>}

accurev merge -io [ -B ] [ -v <ver-spec> | -s <stream> ][--
use_time_basis_for_overlap]

accurev merge [-I <issue-number> [-3] ][--use_time_basis_for_overlap]{ [ -
Fx ] -l <list-file> | <element-list> }
```

Description

The *merge* command edits your workspace's copy of an element by combining its contents with the contents of the version currently in the backing stream. This command can also change the name of the element, if its name in the backing stream differs from its name in your workspace. These two kinds of changes are termed content changes and namespace changes, respectively.

You can specify multiple elements on the command line; each one will be merged as described above.

Variations: you can use the *-v* option to specify another stream's version (instead of the backing stream's version) as the from version; and you can use the *-s* option to merge to a stream other than your workspace stream.

Merge Input

AccuRev uses a 3-way merge algorithm: it considers the from and to versions, along with a third version: the closest common ancestor of these two versions. It determines this common ancestor by analyzing the stream hierarchy and taking into account previous merges for the element. (It does not take into account any *patch* commands you may have executed to incorporate other people's changes into your version.) Thus, if you perform frequent merges on a file, you need to merge only the changes that have occurred since the last merge. You don't have to consider all the changes back to the point at which the two versions diverged.

Performing the Merge

merge reports the three versions being considered, reporting each one in stream-number/version-number format:

```
Current element: ../chap01.doc
most recent ws version: 3/2, merging from: 4/2
common ancestor: 3/1
```

Then, it reports whether a merge of namespace changes is required ...

```
Namespace merge will be required
```

merge

... and whether a merge of content changes is required ...

```
Automatic merge of contents successful. No merge conflicts in contents.
```

```
or
```

```
** merge conflicts in contents, edit needed **
```

Performing the Namespace Merge. If a namespace merge is required, *merge* prompts you to choose one of the names:

```
Resolve name conflict by choosing name from:
```

```
(1) common ancestor: \.\ch01.doc  
(2) backing stream : \.\chapter1.doc  
(3) your workspace : \.\chap01.doc
```

```
Actions: (1-3) (s)kip (a)bort (h)elp
```

As the above example illustrates, a namespace merge is required if you have changed the element's name in your workspace *and* the backing stream version also reflects a name change.

Performing the Content Merge. If a content merge is required, *merge* creates a temporary “merged file”, containing:

- All the changes between the common ancestor and the to version (the file in your workspace).
- All the changes between the common ancestor and the from version (the backing-stream version).

Non-conflicting changes are automatically inserted into the merged file. A change is non-conflicting if *one* of the contributors — either the “from” version or the “to” version — changed the content at that point in the file, but the other contributor didn't make a change. (If exactly the same change was made in both contributors, that's non-conflicting, too.)

A conflicting change occurs where *both* contributors contain a changes from the common ancestor. In this case, *merge* inserts both the “from” and “to” text sections into the merged file, along with separator lines:

```
<<<<<< Your_Version                               (separator: start of conflict section)  
text in "to" version"  
text in "to" version"  
text in "to" version"  
====                                              (separator)  
text at same location in "from" version"  
text at same location in "from" version"  
>>>>>> Backing_Version                           (separator: end of conflict section)
```

After creating the merged file, *merge* asks what you want to do next:

```
Actions: keep, edit, merge, over, diff, diffb, skip, abort, help
```

```
action ?
```

The choices are:

keep Overwrite the to version (the file in your workspace) with the current contents of the merged file; then *keep* a new version of the element in your workspace stream. If you haven't edited the merged file, it may include conflict sections with separator lines, as shown above.

edit Open a text editor on the merged file. Edit the conflict sections to resolve the conflicts and remove the separator lines.

merge

merge Discard the current contents of the merged file and restart the merge process on this file.

over Replace the contents of the merged file with the contents of the to version (the file in your workspace). This is an override, declaring that you prefer your own version to the version in the backing stream.

diff Compare the current contents of the merged file with the to version (the file in your workspace).

diffb Compare the current contents of the merged file with the from version (the version in the backing stream).

skip Cancel the merge for this file, and proceed to the next file specified on the command line (if any).

abort Cancel the merge for this file, and end the *merge* command immediately.

help Display short descriptions of these choices.

If there are conflicts to be resolved, the default is to edit the merged file. If there are no conflicts, the merge is complete, and the default is to keep the merged file.

Choosing to edit the merged file launches a text editor. To resolve the conflicts:

1. Search for the separator lines.
2. In each conflict section, choose the change you would like, combine the changes, or edit the file however you would like.
3. Make sure to remove all the separator lines that mark the conflict section.

After you end the edit session, *merge* prompts you again to take an action. This time, “keep” is the default.

Merging Binary Files

There are no commonly accepted algorithms for merging binary files. Thus, if an overlap occurs in a binary-format element, there are only two ways to resolve the overlap:

- **Use the copy in your workspace.** Execute a *merge* command, and choose to *keep* the result. This keeps the copy in your workspace, and records a merge from the backing-stream version to your newly-kept version.
- **Use the backing stream version.** Execute a *purge* command. This resolves the overlap by removing the element from your workspace stream. Your workspace reverts to having a copy of the backing stream’s version of the element.

Determining Which Elements Need Merging

Use the *stat -o* command to list the elements that need to be merged before you can promote them to the backing stream. This lists all the elements whose changes in your workspace overlap changes in the backing stream.

Using Merges to Implement a Partial Update of Your Workspace

The *update* command performs its work on all the elements in your workspace. Sometimes, however, you may want to update just a selected set of elements to the versions currently in the backing stream. You can use a series of *merge* commands to do so. For example, if your backing stream is named **gizmo_dvt** and

merge

you want to update elements **foo.c**, **bar.c**, and **read.me**, this set of commands accomplishes the “partial update”:

```
accurev merge -v gizmo_dvt foo.c
accurev merge -v gizmo_dvt bar.c
accurev merge -v gizmo_dvt read.me
```

(You’ll probably want to use a script or a loop-processing command to make this task user-friendly.)

User Preferences

- **AC_MERGE_CLI**: An environment variable used by *merge* to invoke an alternative merge program. (The default merge program is *acdif3* in the AccuRev **bin** directory.) The command string must specify all four files involved in the merge, using these substitution patterns:

%a% or **%a** — Filename of the common ancestor version.

%1% or **%1** — Filename of the version in the backing stream (or other non-workspace version).

%2% or **%2** — Filename of the version in the workspace.

%3% or **%3** — Title to be displayed for the common ancestor version.

%4% or **%4** — Title to be displayed for the version in the backing stream (or other non-workspace version).

%5% or **%5** — Title to be displayed for the version in the workspace.

%o% or **%o** — Merge results (output) file.

The values that AccuRev substitutes for the **%3%**, **%4%**, and **%5%** patterns are *not* enclosed in double-quotes, even though the values typically include SPACE characters. The values that AccuRev substitutes for the other patterns *are* enclosed in double-quotes.

Make sure that the merge program you specify is on your search path. The program’s exit status should be:

- Zero if there are no conflicting changes. The *merge* command’s default “do next” action after running this program will be **keep**.
- Non-zero if there are conflicting changes. The *merge* command’s default “do next” action after running this program will be **edit**.

Options

- B** (implies **-i**) Considering the entire backing chain of the workspace’s version, list each version that requires a merge with its parent version. This is similar to *stat -B*, but lists only the versions with **overlap** status.
- i** Display information about merge requirements, but don’t actually merge.
- G** Use AccuRev’s graphical Merge tool to perform the merge if there are conflicts which require human interaction (any value set in **AC_MERGE_CLI** is ignored).
- K** If an overlap with the backing-stream version does not involve any conflicts (and thus, no human intervention is required for the merge), perform the merge and **keep** the new version without asking for confirmation.
- o** Merge all files that need merging (files with **(overlap)** status).

merge

-O For each file to be merged, use the workspace version — as if you had selected **over** when *merge* prompted you to specify a merge action.

-I <issue-number>

Merge the contents of the change package of the specified AccuWork issue record. That is: for each change package entry indicated by an element argument, merge the entry's head version with your workspace version.

-3

Specifies that the *<issue-number>* specified by the **-I** switch is a third-party ITS key rather than an AccuWork issue number.

--use_time_basis_for_overlap

Use the workspace's update level or the stream's time basis to identify overlap instead of using the current time.

When resolving overlap with a backing stream, this option causes AccuRev to merge against the version that was active in the backing stream at the time specified by the workspace's update level or the stream's time basis.

This option has no effect if the user uses the **-v** or **-I** option to merge against a specific version.

-v <ver-spec>

Merge to your workspace from the specified version or stream, instead of from the workspace's backing stream. You cannot use **-v** and **-s** together.

-s <stream>

Uses your workspace to handle overlaps between the specified stream and its parent stream. That is, *merge* behaves as if *<stream>* were your workspace stream. (And this makes sense only if *<stream>* and your workspace stream both contain the same version of the file.) You cannot use **-v** and **-s** together. If you use this option, you must specify elements using depot-relative pathnames.

As with all merges, the results are placed in your workspace: *keep*'ing the results creates a version in your workspace stream, not in the *<stream>* you specified.

You can use a series of *promote*'s to propagate the merge results to the parent of *<stream>*.

-l <list-file>

Process the elements listed in *<list-file>*. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any *<element-list>* specified is silently ignored.

<element-list>

One or more element names, separated by whitespace. If you also specify a *<list-file>* using the **-l** option, this *<element-list>* is silently ignored.

-Fx Signals that *<list-file>* (see the **-l** option) is an XML-format file, not a flat text file. Example:

```
<workspace eolType="1"/>
<element
  type="2"                # 1-directory, 2-text, 3-binary, 4-ptext, 5-element link, or 6-symbolic link
```

merge

```
option1="3"           #parent directory - see below for values
option2="1"           #parent eid
option3="3"           #element name - see note below for values
option4=".\.cc.txt"   #backing name
option5=".\.c"        #ancestor name
option6="14"          #eid of element you are merging
option7="C:\DOCUME~1\rtitle\LOCALS~1\Temp\merge60341.tmp"> #temp file
option8="3"           #link target - see below for values
</element>
```

Options 1, 3, and 8 (which specify how to resolve merge conflicts) can have the following values:

- 1 - Use information from the common ancestor
- 2 - Use information from the backing stream
- 3 - Use information from the workspace

Examples

Merge in the changes to file **foo.c** from the backing stream:

```
> accurev merge foo.c
```

Merge in the changes to file **foo.c** from the maintenance stream:

```
> accurev merge -v gizmo_maint foo.c
```

Specify an alternative merge program:

```
> export AC_MERGE_CLI="fmerge %a %1 %2 %o"           (UNIX Bourne shell)
```

```
> set AC_MERGE_CLI="fmerge %a %1 %2 %o"           (Windows)
```

Extended Example

Assume the contents of the three versions are as follows:

- Common ancestor version — the version that both the backing-stream version and your workspace's version are derived from:

```
int
main( int, char ** )
{
    int a;
    a = 1;
    printf ( "a is %d\n" );
}
```

- Backing stream version:

```
int
main( int, char ** )
{
    unsigned int;
    a = 2;
```

merge

```
    printf ( "a is %d\n" );  
}
```

- Copy in your workspace:

```
int  
main( int, char ** )  
{  
    long int;  
    a = 1;  
    printf ( "The value of a is %d\n" );  
}
```

Resolving Conflicts

The merge will automatically apply both of the non-conflicting changes, the one from the backing version and the one from your version. However, the conflicting change will be flagged and you will need to select the change that you want to preserve:

Merged Version

```
int  
main( int, char ** )  
{  
<<<<<"fmerge %%a %%1 %%2"  
    long int;  
=====  
    unsigned int;  
>>>>>  
    a = 2;  
    printf ( "The value of a is %d\n" );  
}
```

To select the change that you want, simply remove the change bars and the change that you don't want.

After saving the results of your changes, select the [keep](#) option from the menu to keep the changes and save a record of the merge. If you merged elements against the backing stream in order to be able to promote your changes, you can now *promote* the merged elements.

See Also

diff, *keep*, *mergelist*, *patch*, *patchlist*, *stat*

Using a Specific Version of an Element on page 14

mergelist

determine which versions need to be promoted between streams

Usage

```
accurev mergelist -s <from-stream> -S <to-stream> [ -o ] [ -d ]
[ -fl ] [ -fx ] { [ -Fx ] -l <list-file> | <element-list> }
```

Description

The *mergelist* command determines which element versions must be promoted from one stream (the “source” or “from” stream) to another stream (the “destination” or “to” stream) in order to fulfill this synchronization requirement: after the promotions, all changes recorded in the source stream will have been propagated to the destination stream.

Note: for some elements, a *merge* may be required before such a promotion can occur; for other elements, the version in the source stream can be promoted to the destination stream immediately, without requiring a *merge* first.

mergelist merely provides information. It does not perform any actual *merge* or *promote* commands. This command is the “engine” used by the GUI’s *Send to Change Palette* command to populate the Change Palette with a list of versions to be merged/promoted.

Typically, you invoke *mergelist* on two non-workspace streams. But the command works for *any* two streams. A case that’s particularly informative is determining which versions need to be promoted from a workspace stream to its own backing stream:

```
accurev mergelist -s gizmo_3.4_john -S gizmo_3.4
```

This command lists all the versions that are active in the workspace stream **gizmo_3.4_john** — that is, the members of the workspace stream’s default group. By definition, the default group records the set of workspace-stream versions that have not yet been promoted to the backing stream.

The ‘mergelist’ Algorithm

mergelist works on an element-by-element basis. For each element in the source stream, it considers the relationship between:

- the real version associated with the version in the source stream
- the real version associated with the version in the destination stream

This relationship determines whether the source-stream version will be added to the list of versions to be promoted to the destination stream.

Note: non-workspace streams contain virtual versions of elements. Each virtual version is associated with (is a reference to) some real version of the element, which was created in some workspace stream.

mergelist

Here are all the cases:

no version of the element exists in destination stream

The source stream's version is placed on the to-be-promoted list. No merge will be required.

the same real version appears in both in the source and destination streams

No difference, so no promotion required.

the source-stream version was derived from the destination-stream version

The source stream's version is placed on the to-be-promoted list. No merge will be required.

The derivation can be either direct (a series of *keep* commands) or indirect (one or more *merge* commands, along with *keep* commands). Note that *patch* commands don't count in this analysis of a version's derivation.

the destination-stream version was derived from the source-stream version

No promotion is required, because the content of the source-stream version is already present in the destination-stream version.

neither version is derived from the other; both are derived from some common ancestor version

The source stream's version is placed on the to-be-promoted list. A merge will be required to combine the two versions' changes from the common-ancestor version (and possibly resolve conflicts in those changes).

'mergelist' vs. 'diff'

The *mergelist* command is quite similar to the form of the *diff* command that shows the differences between two streams:

```
accurev diff -a -i -v <stream-1> -V <stream-2>
```

These commands differs only in how they handle elements for which "the destination-stream version was derived from the source-stream version":

- The *mergelist* command does not place such elements on the to-be-promoted list, because no promotion is required to get the changes into the destination stream. (The changes are already there.)
- The *diff* command *does* report such elements, because the two streams *do* contain different versions of the element.

Options

-s <from-stream>

The stream to be searched for versions that need to be promoted.

-S <to-stream>

The stream that the versions need to be promoted to.

-o Select only elements whose source-stream and destination-stream versions overlap, requiring a merge.

-d Select only elements that are active in the "from" stream's default group.

-fl Display locations only (no status indicators or version-IDs).

mergelist

-fx Display the results in XML format. Use this option when there is a need to include version information for each element listed in the file.

-l *<list-file>*

Process the elements listed in *<list-file>*. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any *<element-list>* specified is silently ignored.

<element-list>

One or more element names, separated by whitespace. If you also specify a *<list-file>* using the **-l** option, this *<element-list>* is silently ignored.

-Fx Signals that *<list-file>* (see the **-l** option) is an XML-format file, not a flat text file. Example:

```
<elements>
  <e l=".\\file_0001" v="4/1" eid="2"></e>
  <e l=".\\file_0002" v="4/1"></e>
  <e eid="4" v="4/1"></e>
</elements>
```

Examples

Find all the versions that need to be promoted in order to propagate all the changes recorded in stream **gizmo_4.3_maint** to stream **gizmo_test**:

```
> accurev mergelist -s gizmo_4.3_maint -S gizmo_test
```

Considering only the elements listed in file **/tmp/myfiles**, find the versions that need to be promoted to incorporate stream **ColorStar1.3.1**'s changes into stream **ColorStar2.0**:

```
> accurev mergelist -s ColorStar1.3.1 -S ColorStar2.0 -l /tmp/myfiles
```

See Also

diff, *merge*, *patchlist*, *promote*

Using a Specific Version of an Element on page 14

mkdepot

create a new depot

Usage

```
accurev mkdepot -p <depot-name> [ -l <stg-loctn> ] [ -Ci | -Cs ] [ -ke ]
```

Description

The *mkdepot* command creates a new AccuRev depot in a specified directory. It also creates the depot's unique base stream, which can be used as the basis for other streams in the depot (see the [mkstream](#) reference page). See [Entity Names](#) on page 5 for information on naming depots.

The depot's physical location is said to be a new slice of the overall AccuRev repository. To change the location of an existing depot, use the *chslice* command.

Ideally, the files and subdirectories in a slice should be accessible only by the user identity under which the AccuRev Server runs. See [Repository Access Permissions](#) on page 5 of the *AccuRev Administrator's Guide*.

The slice's top-level directory must be located on storage that is local to the machine where the AccuRev Server is running. For more information, see [Storage Layout](#) on page 12 of the *AccuRev Administrator's Guide*.

Case-Sensitivity of Depots

All depots store names of files and directories exactly as they are originally entered (e.g. **WidgetGetCount** or **cmdListAll**). Likewise, AccuRev's CLI and GUI client programs display these names exactly as they were originally entered. The difference is that:

- A case-sensitive depot (created with the **-Cs** option) allows users to create two objects (files or subdirectories) in the same directory, with names that differ only in their case (e.g. **makefile** and **Makefile**).
- A case-insensitive depot (default, or created with **-Ci**) does not allow two objects in the same directory to have names that differ only in their case.

We strongly recommend that you make your depots case-insensitive, for compatibility with Microsoft Windows. Your team might not be using Windows right now, but think about the future. You cannot change an existing depot from case-sensitive to case-insensitive, or vice-versa.

Using 'mkdepot' with a Replicated Repository

New depots can be created only in the master repository, not in a replica repository. If a client using a replica repository issues a *mkdepot* command, an error occurs:

```
Cannot create a new depot on the replica server
```

See [Replication of the AccuRev Repository](#) on page 43 on the *AccuRev Administrator's Guide*.

Removing a Depot

A depot can be removed completely from the repository with the *maintain rmdepot* command. This operation is irreversible! For details, see [Removing a Depot from the AccuRev Repository](#) on page 122 of the *AccuRev Administrator's Guide*.

Options

-p <depot-name>

Specify a name for the new depot. The name may begin with a digit (provided there is at least one non-digit in the name), must not begin with a dot (.), must not include either a slash (/) or a backslash (\) and must not be a name of digits.

-Ci (default) Case insensitive. Use on platforms that are not case sensitive (i.e. Windows) or in mixed environments, such as UNIX/Windows.

-Cs Case sensitive. Use only on platforms that are case sensitive (e.g. UNIX/Linux). But we recommend that you not use this option on any platform.

Note: a depot's **-C** setting cannot be changed with a subsequent *chdepot* command.

-ke Create a depot in which an exclusive file lock is placed on a file when any user, in any workspace, begins active development on it. See [Serial Development through Exclusive File Locking](#) on page 25 of the *AccuRev Concepts Manual*.

Note: a depot's **-k** setting can be changed with a subsequent *chdepot* command.

-l <storage-location>

Specify the full pathname of the directory where the depot's slice is to be created. An error will occur if:

- a file or a non-empty directory already exists at that pathname
- the pathname contains a SPACE character (such as **C:\Program Files\accurev_storage\...**), and is not enclosed in quotes
- the location specified is not on the local machine

Default: create the depot at a default location established at AccuRev installation time, typically <*AccuRev-installation-dir*>/storage/depots.

Examples

Create depot **brass** in the default location:

```
> accurev mkdepot -p brass
```

Create depot **brass** in an alternate location, with exclusive file locking enabled:

```
> accurev mkdepot -p brass -ke -l c:\accstore\brass
```

See Also

chdepot, *chslice*, *mkstream*, *show* (depots)

mkgroup

create a new group of users

Usage

```
accurev mkgroup <group-name>
```

Description

The *mkgroup* command creates a new AccuRev group, with a name up to 96 characters long. Groups enable you to implement security based on development roles, using access control lists (ACLs) maintained with the *setacl* and *lsacl* commands. For instance, you can allow/deny access to particular streams and depots based on membership or non-membership in a group and you can also set up roles such as “developer” or “writer”.

See *Entity Names* on page 5 for information on naming groups. If you rename an existing group with the *chgroup* command, you can then create a new group with the original group name.

Examples

Create an AccuRev group named **eng**:

```
> accurev mkgroup eng
```

See Also

chgroup, *lsacl*, *setacl*, *show* (groups and members)

mkref

create a new reference tree

Usage

```
accurev mkref -r <refree-name> -b <backing-stream> -l <location>
    [ -e <eol-type> ]
```

Description

The **mkref** command creates a reference tree, a directory tree containing copies of all the versions in a stream. A reference tree is very similar to a workspace. You use a reference tree for building, testing, data export, and similar activities; you use a workspace for creation of new versions (source-code development). See [Entity Names](#) on page 5 for information on naming reference trees.

mkref fills the directory tree with read-only files. You can work with these files using such **accurev** commands as **hist** and **diff**. But you cannot use any AccuRev command that would modify these files: **co**, **add**, **defunct**, **keep**, **merge**, **move**, **promote**, **purge**, or **undefunct**.

The only AccuRev commands you can use to modify the files in a reference tree are **update** and **pop**. To do so:

- use **update -r <refree>** to update the reference tree from any location
- use **update** or **pop** after navigating to the reference tree location

If the stream's contents have changed since you created the reference tree, **update** overwrites some files with the new versions. (See [Using a Trigger to Maintain a Reference Tree](#) on page 31 in *AccuRev Technical Notes* for information on how to automate the updating of reference trees.)

You can create any number of reference trees for a given stream, on the same machine or on different machines. You can base a reference tree only on a dynamic stream or a snapshot, not on a workspace.

Note: you cannot create a reference tree with the same name as an existing workspace.

If you rename an existing reference tree with the **chref** command, you can then create a new reference tree with the original name.

Streams and Reference Trees

A stream does not actually contain files — it contains the version-IDs of a set of elements. (More precisely, a stream's specifications enable AccuRev to rapidly construct the list of elements and version-IDs.) **mkref** builds a directory tree on your disk, containing copies of the files indicated by those version-IDs.

For example, a stream named **gizmo_rls2_dvt** (stream #32 in its depot) might contain:

```
gizmo.readme      32/8
src/gizmo.c       1/14
src/cmd.c         1/5
doc/gizmo.doc     1/19
doc/relnotes.doc 32/12
```

Only two elements, **gizmo.readme** and **relnotes.doc**, are under active development in this stream. For all the other elements, this stream uses a version that was promoted to the depot's base stream (stream #1).

mkref creates a directory tree containing five files:

- in the reference tree's top-level directory, a copy of version 8 in the **gizmo_rls2_dvt** stream of element **gizmo.readme**

mkref

- in subdirectory **src**, a copy of version 14 in the base stream of element **gizmo.c**, and a copy of version 5 in the base stream of element **cmd.c**
- in subdirectory **doc**, a copy of version 19 in the base stream of element **gizmo.doc**, and a copy of version 12 in the **gizmo_rls2_dvt** stream of element **relnotes.doc**

Options

-r <refree-name>

Specify the name of the reference tree to be created. The name may begin with a digit (provided there is at least one non-digit in the name), must not begin with a dot (**.**), must not include either a slash (**/**) or a backslash (****) and must not be a name of digits.

-b <backing-stream>

Specify the stream to be the backing stream of the reference tree. You must specify a dynamic stream or a snapshot — you cannot specify a workspace.

-l <location>

Specify the full pathname where the reference tree is to be created.

-e <eol-type>

Specify the line terminator to be used when an AccuRev command (e.g. **update**, **co**) copies a version of a text file from depot storage to the reference tree. (In depot storage, all text files are stored with the AccuRev-standard line terminator, NL.) The <eol-type> can be **d** (default: use the standard line terminator for the client machine's operating system), **u** (use the UNIX/Linux line terminator, NL), or **w** (use the Windows line terminator, CR-NL).

Examples

Create a reference tree named **gizmo_snap1**, based on stream **gizmo**, at a particular location:

```
> accurev mkref -r gizmo_snap1 -b gizmo -l /depot/gizmo/gizmo_snap1
```

See Also

[*mkstream*](#), [*show*](#), [*update*](#)

mkreplica add a depot to a replica repository

Usage

```
accurev mkreplica -p <depot-name> [ -l <depot-slice-location> ]
```

Description

The *mkreplica* command adds the specified depot to the local replica repository's set of depots. This command must be directed to an AccuRev Server process that manages a replica repository; the corresponding master repository must contain the specified depot.

mkreplica copies the depot's metadata from the master repository to the replica repository. It can optionally specify a location (using the *-l* option) for the depot's file storage area. If *-l* is not used, this location defaults to a subdirectory under the replica repository's `.../storage/depots` directory.

Information about which depots are replicated is stored in file `replica_site.xml`, in the local replica repository's `site_slice` directory. *mkreplica* creates a new entry in this file.

mkreplica does not copy the storage files for the versions of the depot's elements to the replica repository. Use the *replica sync* command to synchronize the replica and the master repositories.

Options

-p <depot-name>

The depot to be replicated.

-l <depot-slice-location>

The location where the replicated depot directory ("slice") is to be created. This location must be on a local disk of the machine where the AccuRev Server process runs.

Examples

Add depot **kestrel** to the local replica repository.

```
> accurev mkreplica -p kestrel
```

See Also

replica (sync), *rmreplica*

mkrules set or clear multiple include/exclude rules for a workspace or stream

Usage

```
accurev mkrules -l <file>
```

Description

The *mkrules* command uses an XML-format file to set one or more include/exclude rules for a workspace or stream. The following example demonstrates the structure of the input file:

```
<depot name="velo"/>
  <stream name="xray24_dvt_john"/>
  <comment>set two rules for a workspace</comment>
  <elements>
    <element
      kind="excl"
      location=".\.dir01\sub01"/>
    <element
      kind="incldo"
      location=".\.dir02\sub02"/>
    <element
      kind="incl"
      location=".\.dir01\sub03"
      backingStream="xray24_qa"/>
  </elements>
```

You can create an input file for *mkrules* by revising the output of an *lsrules -fmx* command as follows:

- Remove the top-level tags: **<AcResponse>** and **</AcResponse>**.
- To clear an existing rule, change the value of the **kind** attribute from **incl/incldo/excl** to **clear**.
- The **<depot>** element is required unless you are running this command from within a workspace.
- The **<stream>** element is optional; if you omit it, *mkrules* operates on the current workspace.

Important! If the <stream> element is omitted, mkrules must be run from within a workspace.

- The **<comment>** element is optional; it places a comment string in the AccuRev transaction that *mkrules* creates.

Options

-l <file>

Specifies the location of the XML-format message (document), containing the set of include/exclude rules to be set or cleared.

See Also

clear, excl, incl, incldo, lsrules

mksnap

create a new snapshot

Usage

```
accurev mksnap -s <snapshot> -b <existing-stream> -t <time-spec>
```

Description

The *mksnap* command creates a new snapshot, based on an existing stream. The snapshot contains the same set of element versions that the existing stream contained at a particular point in time. The existing stream on which a snapshot is based might change, through *promotes*. But the contents of a snapshot never change. See *Entity Names* on page 5 for information on naming snapshots.

You can implement a “changeable snapshot” facility using a dynamic stream. See [Using the -t Option](#) in the *mkstream* reference page.

Options

-s <snapshot>

(required) Specify a name for the new snapshot. The name may begin with a digit (provided there is at least one non-digit in the name), must not begin with a dot (.), must not include either a slash (/) or a backslash (\) and must not be a name of digits only.

-b <backing-stream>

Specify an existing stream, on which the new snapshot will be based.

-t <time-spec>

Snapshot time. The snapshot will contain the versions that were in the stream at the specified time. If you specify a transaction number, the snapshot will include the versions that were in the stream at the time the transaction was completed.

A time-spec can be any of the following:

- Time in <"YYYY/MM/DD HH:MM:SS"> format, "2007/08/07 20:27:15", for example. Note that you must use quotes when specifying time-spec as YYYY/MM/DD HH:MM:SS to ensure that the entire argument is interpreted by the command shell as a single token.
- Time keyword: **now**
- Transaction number as a positive integer: e.g. *146* or *23965*
- Transaction number keyword: **highest**

You cannot specify a time that is before the creation of the existing stream, or after the current time.

When using the *-t* option, take into account the possibility of a timewarp. The AC_SYNC environment variable (see [AccuRev User Preferences](#) on page 6) determines how a timewarp will be handled. See also section [System Clock Synchronization](#) on page 31 of the *AccuRev Administrator's Guide*.

mksnap

Examples

Create a snapshot of the current state of the **gizmo** stream, calling it **gizmo_s1**:

```
> accurev mksnap -s gizmo_s1 -b gizmo -t now
```

Create a snapshot of the state of the **gizmo** stream on May 2nd, 2000 at 2:55pm, calling it **gizmo_may2**:

```
> accurev mksnap -s gizmo_may2 -b gizmo -t "2000/05/02 14:55:00"
```

See Also

mkstream, *promote*

mkstream

create a new stream

Usage

```
accurev mkstream -s <stream> -b <backing-stream> [ -kp ] [ --gated ]
[ -t <time-spec> ]
```

Description

The *mkstream* command creates a new stream that is based on an existing stream called the backing stream. A stream and its backing stream are sometimes referred to as "child" and "parent" streams, respectively.

You can use the *mkstream* command to create dynamic, pass-through, and gated streams. Use the *mksnap* command to create snapshot streams. See [Entity Names](#) for information on naming streams.

Dynamic Streams

A *dynamic stream* is a stream whose content changes over time, with new versions promoted from child workspaces and/or from other dynamic streams. It also inherits versions from its parent stream. AccuRev creates streams as dynamic streams by default.

Pass-Through Streams

A *pass-through stream* is a special kind of dynamic stream, so called because a version that is promoted to such a stream automatically passes through to its parent stream. The file does not become active in the pass-through stream (that is, the pass-through stream has no default group); it *does* become active in the parent of the pass-through stream.

Pass-through streams are useful for grouping lower-level streams and/or workspaces. This makes it possible to reparent all the streams in the group in a single operation.

You cannot set a basis time on a pass-through stream.

Gated Streams

A *gated stream* is a stream that you use to control whether changes promoted into it are automatically promoted out of it, often based on meeting externally established criteria (such as passing a test suite or completing a successful nightly build, for example). Gated streams combine attributes of both dynamic and pass-through streams, and they have features that are unique to them. For example, like dynamic streams, the content of a gated stream changes over time; and like pass-through streams, versions promoted to a gated stream pass through it to the gated stream's parent. Unique to gated streams is a special type of stream called a *staging stream*.

You cannot set a basis time on a gated stream.

See Chapter 8, "Using Streams to Support Continuous Integration" in the *AccuRev Administrator's Guide* for more information on gated streams.

How Streams Change

A newly created stream contains the same set of element versions as its backing stream. (If you use the *-t* option when creating a dynamic stream, this is the set of versions that the backing stream contained at a particular point in time.) Thereafter, the dynamic stream's contents can change, either "actively" or "passively".

Active Changes to a Stream

The *promote* command places a new version of one or more elements in a stream. This is termed an “active change”, because it is the result of an explicit command (*promote*), and is recorded in the database as a transaction.

Promoting versions of elements into a stream makes those elements “active” in the stream. (A stream’s default group is its set of “active” elements.) Those elements remain active in the stream until the versions are promoted out of the stream, or are purged from the stream.

Promote behaves differently for pass-through and gated streams:

- If you promote versions to a pass-through stream, the versions are actually (and automatically) promoted to the stream’s parent. A pass-through stream cannot have any active elements; its default group is always empty.
- When you promote from a workspace or child stream of a gated stream, those versions become active in a *staging stream* -- an intermediate stream that AccuRev creates automatically when you create or reparent a workspace or stream off it. Changes promoted to a gated stream remain in the staging stream’s default group until they satisfy the criteria established on the gated stream; once they do, those versions are promoted automatically through the gated stream directly to the gated stream’s backing (parent) stream.

A stream’s active elements do not experience any of the “passive” changes described in the next section.

Passive Changes to a Stream

Elements that are not currently active can still change, “passively”. Dynamic, pass-through, gated, and staging streams inherit changes automatically from their backing streams. If a new version of **foo.c** is promoted to a backing (parent) stream, this new version automatically appears in all of the child streams *where foo.c is not currently active*.

Such changes are termed “passive”, because they are implicit and automatic. They are not recorded as separate transactions. Passive changes can only occur to a stream whose parent is, itself, changeable. If the parent is a snapshot, the child won’t experience any passive changes.

Using the -t Option

In effect, the *-t* option “takes a snapshot” of the parent stream at a particular time, called the basis time. But the following are only similar, not quite identical:

- A child, created with *-t <time-spec>*, with a dynamic stream as its parent.
- A child whose parent is snapshot that was defined with the same *<time-spec>*.

Both child streams initially get the same set of versions. And both child streams can *keep* new versions of elements. The difference is in promoting the new versions:

- In a child created with *mkstream -t*, you can *promote* versions to the parent (because the parent is a dynamic stream). The elements remain active in the child stream.

(In other situations, elements become inactive in the child stream after a promotion. Such elements revert to “passively” using the parent stream’s version. In this situation, becoming inactive would make the elements revert to the version that existed at the time specified with *mkstream -t*. This “surprise” is avoided by having the elements remain active.)

- In a child of a snapshot, you cannot *promote* changes to the parent (because the parent is a snapshot, and thus immutable). You can propagate the changes to other dynamic streams using *promote -s -S* or the GUI’s Change Palette. This operation leaves the changed files active in the child stream.

mkstream

You may wish to avoid the complications described above by not using **mkstream -t** to create streams for active development. Instead, you can use **mkstream -t** to create “changeable snapshot” streams:

- Create a child stream using **mkstream -t <time-1>**, and prevent changes to the stream using **lock**. This is similar to creating a snapshot of the parent stream at **<time-1>**.
- Subsequently, after changes have been made in the parent stream, unlock the child stream, update its “snapshot” with **chstream -t <time-2>**, then lock it again.

Options

-s <stream>

(required) Specify a name for the new stream. The name may begin with a digit (provided there is at least one non-digit in the name), must not begin with a dot (**.**), must not include either a slash (**/**) or a backslash (****) and must not be a name of digits only.

-b <backing-stream>

Specify an existing stream, on which the new stream will be based.

-kp Make the new stream a pass-through stream.

--gated Make the new stream a gated stream.

-t <time-spec>

Bases the new stream on the state of the backing stream at the specified basis time. A time-spec can be any of the following:

- Time in **<"YYYY/MM/DD HH:MM:SS">** format, **"2007/08/07 20:27:15"**, for example. Note that you must use quotes when specifying time-spec as **YYYY/MM/DD HH:MM:SS** to ensure that the entire argument is interpreted by the command shell as a single token.
- Time keyword: **now**
- Transaction number as a positive integer: e.g. **146** or **23965**
- Transaction number keyword: **highest**

You cannot specify a time that is before the creation of the backing stream, or after the current time. You cannot set a basis time on a pass-through stream.

See [Using the -t Option](#) above.

When using the **-t** option, take into account the possibility of a timewarp. The **AC_SYNC** environment variable (see [AccuRev User Preferences](#)) determines how a timewarp will be handled. See also section [System Clock Synchronization](#) on page 31 of the *AccuRev Administrator's Guide*.

Examples

Create a dynamic stream backed by another dynamic stream:

```
> accurev mkstream -s gizmo_maint -b gizmo
```

Create a stream based on **gizmo** at the present time:

```
> accurev mkstream -s gizmo_fixes -b gizmo -t now
```

mkstream

Create a changeable snapshot of the **gizmo** stream at May 2nd, 2000 at 2:55pm:

```
> accurev mkstream -s gizmo_s1 -b gizmo -t "2000/05/02 14:55:00"  
> accurev lock gizmo_s1
```

Create a gated stream called **gizmo_gate** backed by stream **gizmo**:

```
> accurev mkstream -s gizmo_gate -b gizmo --gated
```

See Also

chstream, lock, promote, remove, unlock

mktrig

activate a trigger in a depot

Usage

```
accurev mktrig [ -p <depot-name> ] [ pre-create-trig | pre-keep-trig |
pre-promote-trig | server-post-promote-trig ] <program-or-script>
```

Description

The *mktrig* command creates a trigger within a particular depot. A trigger can “fire” before or after certain AccuRev transactions occur. The following triggers are defined on a per-depot basis, affecting operations only with that particular depot:

pre-create-trig

fire on the client machine, before an *add* transaction

pre-keep-trig

fire on the client machine, before a *keep* transaction

pre-promote-trig

fire on the client machine, before a *promote* transaction

server-post-promote-trig

fire on the server machine, after a *promote* transaction

When a trigger fires, it executes the trigger program or trigger script that you specify in the *mktrig* command. The names of all elements involved in the transaction are passed to this program or script as part of a parameters file.

mktrig does not make a copy of the trigger script. It simply records the location at where AccuRev will look for the script when the trigger fires.

Note: The trigger script need not exist at the specified location when you issue the *mktrig* command. AccuRev doesn’t look for the script until the trigger fires.

A trigger fires exactly once per transaction. If you do a *keep* operation on 10 elements and there is a pre-keep trigger set, the trigger script will execute once, with an argument list containing all 10 element names. It is the script’s responsibility to process its entire argument list.

By acting once for the entire transaction, the script can abort a transaction if there is a problem with any element in the transaction, or you can send out a single email message with a list of all of the elements, instead of a separate message for each element in the transaction. For a transaction with 100 elements, that’s a real benefit!

When the trigger script executes, its exit value should be 0 (success) or non-zero (failure). When a pre-operation trigger script fails, execution of the AccuRev operation is canceled.

Location of the Trigger Program or Script

The trigger script must be installed in a place that is accessible to everyone that will be using a depot with the trigger enabled. Separate copies of the script can be placed on each client machine. A better procedure is to place all scripts in a central, network-wide location that all client machines can “see” (UNIX/Linux mount or Windows network share). If triggers will fire on both Windows and UNIX/Linux machines, be

mktrig

sure to read [Notes on Triggers in Multiple-Platform Environments](#) on page 81 of the *AccuRev Administrator's Guide*.

When a trigger fires, AccuRev looks for a script at the location you specify in the **mktrig** command. A pre-transaction trigger fires on the client machine:

- AccuRev searches for the trigger script on the client machine, in the directories on the user's search path.
- The script executes on the client machine, with the user's identity.

A post-transaction trigger fires on the server machine:

- AccuRev searches for the trigger script on the server machine, using the search path of the "server account": the user account that runs the **accurev_server** process — typically, **acserver** (UNIX/Linux) or **System** (Windows).
- The script executes on the server machine, with the server-account identity.

Note: AccuRev uses a search path to locate the trigger script if you specify a simple name or a relative pathname for the script in the **mktrig** command (this is our recommendation); if you specify an absolute pathname for the script, no search path is involved.

The 'server_preop_trig' and 'server_admin_trig' Trigger Scripts

The triggers described above operate on a per-depot basis. For example, a **pre-create-trig** trigger might be activated in depot **gizmo**, but not in depot **whammo**. Two additional triggers execute on the AccuRev server machine:

- The **server_preop_trig** script executes before the command is executed, and can cancel the command altogether. This is a per-depot trigger.
- The **server_admin_trig** script executes after the command is executed. This script cannot cancel execution of the command. This trigger applies to AccuRev commands irrespective of which depot, if any, is being addressed. Note that the **server_admin_trig** script applies to *all* depots.

You enable these triggers not with the **mktrig** command, but by installing the trigger scripts at well-known locations. For details, see [Server-Side Triggers](#) on page 77 of the *AccuRev Administrator's Guide*.

AccuWork Triggers

AccuRev's issue management facility, AccuWork, also supports triggers. See [server_dispatch_post](#) on page 81 of the *AccuRev Administrator's Guide*.

Options

-p <depot-name>

Specify the depot in which the trigger is to be enabled (default: the depot of the current workspace).

Examples

Execute Perl script **elem-type.pl** before each **add** transaction:

```
> accurev mktrig pre-create-trig /usr/local/triggers/elem-type.pl
```

Execute Perl script **addheader.pl** before each **keep** transaction:

```
> accurev mktrig pre-keep-trig /usr/local/triggers/addheader.pl
```

mktrig

See Also

add, keep, rmtrig, show (triggers)

[AccuRev Triggers](#) on page 77 of the *AccuRev Administrator's Guide*.

mkuser

register a new username

Usage

```
accurev mkuser { -kf | -kd | -kk } <principal-name> [ <password> ]
[ -e <email_address> ] [ -s <ssh_public_key> ]
```

Description

The *mkuser* command defines a new user by creating a new principal-name in AccuRev's user registry. The user is licensed in one of these ways:

- can use AccuRev configuration management, AccuWork issue management, and GitCentric (“full” license)
- can use GitCentric and AccuWork issue management only
- can use AccuWork issue management only

AccuRev operations can be performed only by someone who has a principal-name in the registry, and has a license appropriate for the operation. Each transaction records the principal-name of the person who performed it.

A principal name must start with a letter (a-z, A-Z); it can contain only letters, underscores, spaces, hyphens, periods, plus signs, the commercial at sign (@), and numbers.

To list the existing principal-names, use the *show users* command. To rename a user (e.g. to fix a spelling error or change **cindi** to **cyndee**), or to change the kind of license for a user, invoke the *chuser* command.

If you rename an existing user with the *chuser* command, you can then create a new user with the original principal-name. The two users have different numeric user-IDs, and are unrelated.

Principal-names and Usernames

Your principal-name is independent of the username (login name) by which the operating system knows you. You might be working in a heterogeneous network, where there are multiple incompatible operating-system user registries. Similarly, your organization might be using AccuRev at multiple sites. In such situations, you can (and must!) use the same principal-name wherever you work, even though your operating-system username varies from machine to machine.

Security: Principal-names and Passwords

Principal-names are also needed for AccuRev security. A user with a password is considered to be an “authorized user” by the AccuRev ACL facility.

Options

-e <email_address>

Allows you to specify an email address for the user. The address length is limited to 256 characters.

-kf (default) Create a user licensed for AccuRev, AccuWork, and GitCentric.

-kd Create a user licensed for AccuWork only. The user can access AccuWork issues in a depot (unless restricted by an ACL permission), but cannot access the version-controlled files in any depot.

mkuser

-kk Create a user licensed for both GitCentric and AccuWork

-s <*ssh_public_key*>

Allows you to specify an SSH public key for the user. When specifying the **-s** option:

- Enter text characters only. If you are copying the SSH key from an **id-rsa.pub** file, for example, make sure not to introduce new-line characters or spaces.
- You can omit the **ssh-rsa** string that appears before the key, and the email address that appears at the end of the key. If you choose to include these values, however, the entire argument for the **-s** option must be in double quotes. For example:
"ssh-rsa AAAAB3NzaC1yc2EAAAADA.../tvfvdf017me/wh user@company.com"

Note that ssh-rsa is assumed. If you are using ssh-dss, you must include the entire argument as shown in the preceding example, substituting **ssh-dss** for **ssh-rsa**.

A general discussion about the SSH protocol is beyond the scope of this document, so use a search engine with terms like 'SSH', and 'tutorial' for more information.

Examples

Create a new user named **john_smith**, licensed to use both the configuration management and issue management facilities.

```
> accurev mkuser john_smith
```

Create a new AccuWork user named **jane_doe**, with password **cv78w**.

```
> accurev mkuser -kd jane_doe cv78w
```

See Also

chpasswd, *chuser*, *lsacl*, *setacl*, *show* (users)

mkws

create a new user workspace

Usage

```
accurev mkws -w <workspace-name> -b <backing-stream> -l <location>
    [ -k <kind> ] [ -e <eol-type> ] [ -i ]
```

Description

The *mkws* command creates a workspace, in which you can work with the version-controlled files (elements) in a particular AccuRev depot. See *Entity Names* on page 5 for information on naming workspaces.

A workspace consists of two parts:

- Workspace tree: a directory tree located in your machine’s disk storage, or in remote storage accessed through a network file system. The directory tree is simply a collection of files that is (loosely speaking) your copy of the depot.
- Workspace stream: a stream in the designated depot, dedicated to the workspace. All changes you make in the workspace are initially recorded in this “private” stream.

You use *accurev* commands to move development data between the workspace tree (temporary private storage) and the depot (permanent public storage):

- The *pop* (“populate”) and *update* commands copy files from the depot to the workspace tree. Each file is a copy of a particular version of some element.
- The *keep* command copies files from the workspace tree to the depot. Each copied file becomes a new version of the element, in the workspace stream. The creation of the new version is recorded as a transaction in the database.

When you create a workspace, you assign it a simple workspace name. The associated workspace stream is assigned the same name, because it is dedicated to the workspace. This name must be unique across all depots; *mkws* helps you out in this regard by automatically appending your principal-name to the name you specify with the *-w* option.

Note: You cannot create a workspace with the same name as an existing reference tree.

Initial Update Level

Each workspace has an update level, which indicates how up-to-date the workspace’s files are. For example, an update level of 3475 indicates that your workspace has incorporated (through the *update* command) changes recorded in transactions up to and including transaction #3475.

When you create a workspace, *mkws* does not copy any files into the workspace tree. The workspace’s update level is initially set to transaction #0. Since no version of any element existed at this update level (before any transactions were recorded in this depot), there is no data to be copied into the workspace. The *-i* option (see below) provides a variant of this workspace initialization scheme.

Note: When you create a new workspace in the AccuRev GUI, the update level is set to the most recent transaction, and files *are* copied into the workspace tree.

To fill a new workspace with files, enter an *update* command. This creates a directory hierarchy in the workspace tree and sets the update level to the depot’s most recent transaction.

Location of the Workspace Tree in the File System

You can create a workspace tree anywhere in the client machine's file system — subject to operating-system access controls. But don't create a workspace within the AccuRev installation area: typically, **C:\Program Files\AccuRev** (Windows) or **/opt/accurev** (UNIX/Linux).

You can also create a workspace on a remote machine's disk, as long as that location is accessible on your machine through the operating system's remote-file-access capability ("sharing" on Windows systems, "mounting" on UNIX/Linux systems). Again, access control imposed by the network file system may restrict where you can create a workspace.

Note: The workspace tree must be fully contained within a single physical file system, whether local or remote. Example: you create a workspace at location **/opt/workspaces/derek**; an error occurs if a remote file system is mounted at **/opt/workspaces/derek/src**.

Workspace Options

By default, **mkws** creates a standard workspace, which has the following characteristics:

- It contains all available elements. That is, the workspace contains a version of every element in the workspace's backing stream. (You must follow the **mkws** command with an **update** command to perform an initial load of these versions into the workspace tree.)

After creating the workspace, you can control exactly which elements appear in it, using the include/exclude facility. This facility is implemented through the commands **incl**, **incldo**, **excl**, **clear**, and **lsrules**. (You don't have to specify any **mkws** option to enable the include/exclude facility. It's always enabled for all workspaces.)

- The version-controlled files in the workspace tree are writable at any time. There is no need to perform a checkout operation on a file before beginning to work on it.

You can override this characteristic in either of these ways: having the workspace use the exclusive file locking feature (**-ke** option), or having the workspace use the anchor required feature (**-ka**). These features are similar: they both require you to checkout a file, using the **anchor** or **co** command, before modifying it. Exclusive file locking goes further, preventing multiple users from working on the same file at the same time. See [Serial Development through Exclusive File Locking](#) on page 25 of the *AccuRev Concepts Manual*.

An exclusive file locking workspace cannot subsequently be changed with the **chws** command. This restriction also applies to a workspace that was not itself created with the **-ke** option, but whose depot does use the exclusive file locking feature.

- Version-controlled text files in the workspace tree use the line terminator appropriate for your machine's operating system: NL for UNIX/Linux, CR-LF for Windows.

You can override this characteristic with the **-e** option: **-eu** forces the workspace's version-controlled text files to use the UNIX/Linux line terminator; **-ew** forces the workspace's version-controlled text files to use the Windows line terminator.

Workspace Flexibility and Persistence

The naming scheme separates a workspace's logical name from its physical location in the file system. This makes it easy to move a workspace to a larger disk, to a faster machine, to a disk farm, etc. You must use the **chws** command to inform AccuRev of such a location change.

If you create a workspace by mistake, you can make it invisible with the **rmws** command. This does not actually delete the workspace — that would violate AccuRev's TimeSafe property. Once created, a

workspace persists in the workspace registry forever. *rmws* merely hides the workspace, and attempts to create another workspace with the same name will fail.

Converting an Existing Directory Tree into a Workspace

The location you specify with the *-I* option can be an existing directory. This has the effect of turning the entire directory tree at that location into a new workspace. All the files and subdirectories in the directory tree start out as external objects. To convert them all to version-controlled file and directory elements, execute the command *accurev add -x* from anywhere within the newly created workspace.

Be careful to avoid the mistake of trying to “convert the same tree twice”. For example, here's an incorrect usage scenario:

1. In the pre-AccuRev era, John and Mary each made a copy of the organization's “official” source tree. They each started to modify a few files within their personal copy of the tree.
2. The organization adopts AccuRev, and John and Mary install the software on their machines.
3. John converts his copy of the source tree to a workspace.
4. Mary converts her copy of the source tree to a workspace.

Everything seems OK, but name collisions will occur when both John and Mary promote versions of their newly created file elements to the common backing stream. For example, Mary might promote her version of file **lib/gizmo.h** to the backing stream first. When John subsequently attempts to promote his version of **lib/gizmo.h** to the same backing stream, AccuRev will complain “Name already exists in backing stream”.

What's the correct procedure? Only one person should create AccuRev elements from an existing source tree. If John has already performed step #3, they should proceed as follows:

1. John promotes all his newly created elements to the common backing stream, using the command *accurev promote -d*.
2. Mary creates a new, empty workspace with *mkws*, and uses the *update* command to load John's versions into her workspace.
3. Mary selectively copies the files that she modified from her copy of the original source tree to her new workspace. She updates the timestamps on these file with *accurev touch*.

Options

-w <workspace-name>

Specify a name for the new workspace. The name may begin with a digit (provided there is at least one non-digit in the name), must not begin with a dot (*.*), must not include either a slash (*/*) or a backslash (**) and must not be a name of digits only. *mkws* adds the suffix *_principal-name* to the name you specify (if you don't include the suffix yourself).

-k <kind>

Specify the kind of workspace to be created.

- *-kd*: (“default”) Create a workspace that doesn't have any of the features enabled by the other *-k* options.
- *-ke*: Create a workspace in which an exclusive file lock is placed on a file when you begin active development on it. See [Serial Development through Exclusive File Locking](#) on page 25 of the *AccuRev Concepts Manual*.
- *-ka*: Create an anchor-required workspace, in which you must use the *anchor* or *co* command on a file before modifying it.

mkws

The *-ke* and *-ka* options are mutually exclusive.

See the [incl](#) and [excl](#) reference pages for a way to configure a workspace with just the files you need. Also, see the *-i* option below.

-b <backing-stream>

Specify the stream to be the backing stream of the workspace.

-l <location>

Specify the full pathname where the workspace is to be created.

-e <eol-type>

Specify the line terminator to be used when an AccuRev command (e.g. *update*, *co*) copies a version of a text file from depot storage to the workspace. (In depot storage, all text files are stored with the AccuRev-standard line terminator, NL.) The <eol-type> can be *d* (default: use the standard line terminator for the client machine's operating system), *u* (use the UNIX/Linux line terminator, NL), or *w* (use the Windows line terminator, CR-NL).

-i

Configure the workspace to be empty initially, by implicitly executing the command *incldo* on the top-level directory of the depot. The new workspace is also updated to the depot's most recent transaction.

Examples

Create a workspace for bugfixes in `/home/jsmith/ws`, backed by the **gizmo1.1** stream:

```
> accurev mkws -w gizmo_bugfix -b gizmo1.1 -l /home/jsmith/ws/test
```

This creates a workspace and a stream named **gizmo_bugfix_john_smith**. (if your principal name is **john_smith**).

See Also

[excl](#), [incl](#), [mkref](#), [show](#) (streams), [start](#)

move

move or rename elements

Usage

```
accurev move [ -c <comment> ] { <file-element-name> | -e <eid> } <new-name>

accurev move [ -c <comment> ] { <file-element-name> | -e <eid> }
    <destination-directory>

accurev move [ -c <comment> ] { <file-element-name> | -e <eid> }
    <destination-directory>/<new-name>

accurev move [ -c <comment> ] { <directory-element-name> | -e <eid> |
    -l <list-file> } <destination-directory>
```

Description

Note: You can spell this command name either *move*, *mv*, or *rename*.

The *move* command changes the pathname at which an element is accessed. You can use this command to perform a simple renaming, to move an element to another directory, or both. You cannot move an element to another depot. (This would violate AccuRev's TimeSafe property: the element would magically appear in the past of the destination depot.)

You must specify a single element to move, either by name or by element-ID (*-e* option). Specifying a directory makes it a subdirectory of the *<destination-directory>* you specify.

Note: You can create a new directory with this command. The new directory is created as part of the *move* transaction.

Exclusive File Locking and Anchor-Required Workspaces

See *File Locking in Workspaces* on page 5.

Options

-c <comment>

Specify a comment for the transaction. The next command-line argument should be a quoted string. Alternatively, the next argument can be in the form *@<comment-file>*, which uses the contents of text-file *<comment-file>* as the comment.

-e <eid>

The element-ID of the element to be moved. If you also specify a *<list-file>* or *<element-list>*, it is ignored.

-l <list-file>

Process the elements listed in *<list-file>*. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in a *<list-file>*.

If you use this option, you cannot also use the *-e* option.

move

Examples

Rename file **foo.c** to **bar.c**:

```
> accurev move foo.c bar.c
```

Rename directory **lib** to **library**:

```
> accurev move lib library
```

Move file **foo.c** to the parent directory:

```
> accurev move foo.c ..
```

Move file **foo.c** into subdirectory **subdir**, and rename it to **bar.c**. (This command will create **subdir** if it did not already exist.):

```
> accurev move foo.c subdir/bar.c
```

See Also

[promote](#)

[Techniques for Selecting Elements](#) on page 10

name

name list the name of the element with the specified element-ID

Usage

```
accurev name [ -v <ver-spec> ] [ -p <depot-name> ] [ -fv | -fx ] -e <eid>
```

Description

The **name** command displays the name of an element, given its integer element-ID. By default, **name** displays the name that the element has in your workspace stream. With the **-v** option, you can display the element's name in another stream.

Double Vision: Seeing an Element Multiple Times in a Workspace

One consequence of AccuRev's cross-link facility is that two (or more) different versions of the same element can appear at different pathnames in the same workspace or stream. This is called double vision. It is not an error — at least, not from AccuRev's perspective. Seeing the same element twice might be exactly what you intended, or it might signify that you've left some refactoring work unfinished.

In a double-vision situation, the **name** command lists all of an element's pathnames in a workspace or stream:

```
> accurev name -e 28 -v topaz_refact
  \.\tools\tools.readme
  \.\README-tools.txt
```

You can use the **-fv** option to display the cross-link details in a double-vision situation:

```
> accurev name -fv -e 305
  \.\dir09\sub03\file04.txt
  \.\dir02\sub02\moved_from_9_3_4.txt
  xlink \.\dir02 from amen55_mnt_john to amen55_dvt
```

Options

-e <eid>

The element-ID of the desired element.

-v <ver-spec>

Display the name of a particular version of the element, instead of the version in your workspace stream. See [Using a Specific Version of an Element](#) on page 14 for a description of the forms that **<ver-spec>** can take.

-p <depot-name>

The depot in which to look for the element. (default: the depot of the current workspace).

-fx Display the results in XML format. This includes cross-link information, if any (see **-fv**).

-fv For each version that appears in the workspace or stream through a cross-link, display that cross-link:

```
xlink \.\dir02 from amen55_mnt_john to amen55_dvt
```

name

Examples

Display the name in your workspace stream of the element with element-ID 311:

```
> accurev name -e 311
```

Display the name that stream **gizmo_dvt** currently uses for the element with element-ID 311:

```
> accurev name -v gizmo_dvt -e 311
```

See Also

move

patch incorporate a set of changes from a given workspace into the current version

Usage

```
accurev patch [ -G ] [ -O ] [ -j <ver-spec> ] -v <ver-spec>
    { <element-list> | [ --from_stream=streamname ] -l <list-file> }

accurev patch [ -G ] [ -O ] [ -I <issue-number> [ -3 ] ]
    { <element-list> | -l <list-file> }
```

Description

The *patch* command is a variant of *merge*:

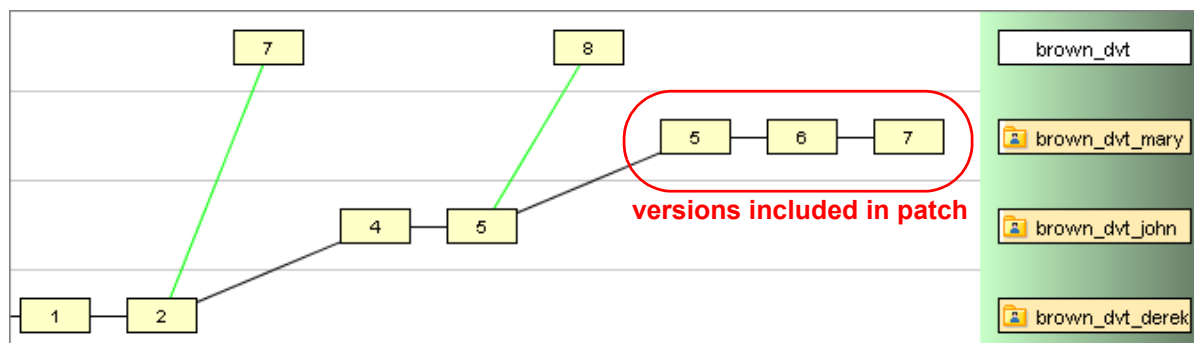
- *merge* incorporates the entire set of changes between the from version and the common ancestor version.
- *patch* incorporates just the set of “recent changes” in the from version. It determines this set of changes by starting at the from version and scanning backward through the file’s ancestry. The scan stops when it encounters a version that was originally created in another workspace, or a version that was promoted to another stream. This older version, termed the basis version, is judged to precede (and not belong to) the set of “recent changes” in the from version (which is also termed the head version).

You can use the *-j* option to suppress this automatic determination of the basis version, specifying a particular version to be used as the basis version.

You can specify multiple elements on the command line; each one will be patched as described above.

Basis Version Examples

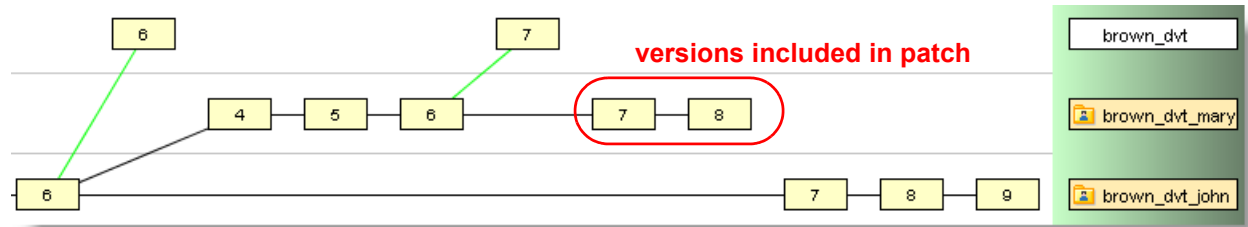
Example 1: Before Mary started her recent work, she updated her workspace. This brought in a version of the file originally created by another user — say, version **brown_dvt_john/5**. Then she proceeded to create versions 5 through 7 in workspace **brown_dvt_mary**.



Derek decides to patch version **brown_dvt_mary/7** into his workspace. AccuRev searches backward through the element’s ancestry, and includes the set of changes recently made in Mary’s workspace: this set includes **brown_dvt_mary/5** through **brown_dvt_mary/7**; the patch doesn’t include the two versions created in workspace **brown_dvt_john**.

patch

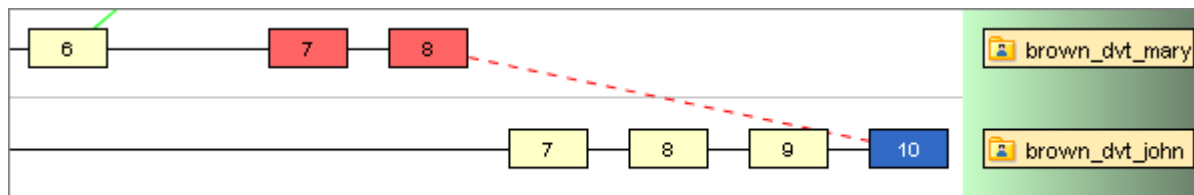
Example 2: Suppose Mary started by bringing version **brown_dvt_john/6** into her workspace with an update. Then she created versions **brown_dvt_mary/4** through **brown_dvt_mary/6**, promoted her work to the backing stream, and created two more versions: **brown_dvt_mary/7** and **brown_dvt_mary/8**.



When John patches version **brown_dvt_mary/8** into his workspace, AccuRev decides that only the versions since the promotion — versions 7 and 8 — contain “recent changes”. The idea is that a promotion typically marks the end of a programming task, not an intermediate checkpoint.

Indicating a Patch and its Set of Versions

The Version Browser uses a dashed red line to indicate a version created by a *Patch* command. Selecting the version causes the set of versions included in the patch to be highlighted in red.



The basis version is recorded in the *keep* transaction that records a *Patch* command in the repository:

```
transaction 206; keep; 2005/12/29 15:57:57 ; user: john
# patched
\\.dir00\sub02\file03.txt 5/10 (5/10)
ancestor: (5/9) patched from: (4/8)-(4/6)
```

In this command, version 4/8 (**brown_dvt_mary/8**) was patched into version 5/9 (**brown_dvt_john/9**), and the result was kept as version 5/10 (**brown_dvt_john/10**). The backward search through the file’s ancestry identified 4/6 (**brown_dvt_mary/6**) as the basis version.

Options

- G** Use the graphical Merge tool to perform the operation if there are any conflicts (which require human interaction).
- O** When merging each file, use the workspace version’s contents.
- I <issue-number>**
Patch the contents of the change package of the specified AccuWork issue record. That is: for each change package entry indicated by an element argument, patch the entry’s head version into your workspace version.
- 3**
Specifies that the <issue-number> specified by the **-I** switch is a third-party ITS key rather than an AccuWork issue number.

patch

--from_stream=<stream_name>

Displays output all segments of visible variants of the specified issue in the specified stream. For discussions about segments and variants, see the "Change Packages" chapter of the AccuRev *On-Line Help*.

-j <ver-spec>

Specify the basis version. Must be used with **-v <ver-spec>**.

-l <list-file>

Process the elements listed in *<list-file>*. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any *<element-list>* specified is silently ignored.

<element-list>

One or more element names, separated by whitespace. If you also specify a *<list-file>* using the **-l** option, this *<element-list>* is silently ignored.

-v <ver-spec>

Specify the version whose changes are to be incorporated into your workspace's copy of the element. The corresponding basis version is determined automatically if you do not use the **-j** option.

Examples

Patch in the change that was finished in version 4/3 of **foo.c**:

```
> accurev patch -v 4/3 foo.c
```

Patch in the change that was finished in version 5/19 of **foo.c**, using version 4/6 as the basis version:

```
> accurev patch -v 5/19 -j 4/6 foo.c
```

See Also

[anc](#), [diff](#), [merge](#), [mergelist](#), [patchlist](#)

[Using a Specific Version of an Element](#) on page 14

patchlist list versions that need to be patched into the workspace's version

Usage

```
accurev patchlist -v <from-ver-spec> [ -V <to-ver-spec> ]
    [ -t <transaction-range> ] [ -fx ] [ <element-list> | -e <eid> ]
```

Description

The *patchlist* command compares two versions of each specified element: the “from” version specified with *-v* and the “to” version specified with *-V*. If you omit the *-V* option, the version in the current workspace is used; if you don’t specify any elements, all the elements in the “to” workspace/stream are considered.

If a transaction range is specified with *-t*, *patchlist* only considers the versions involved in those transactions for the comparison.

For each element, *patchlist* lists all the real versions that contain changes that are present in the “from” version, but have not yet been incorporated into the “to” version. If no elements are specified, all the elements in the “to” workspace/stream are considered.

For example, to find which of Allison’s versions of file **brass.h** contains changes that Derek has not yet incorporated:

```
accurev patchlist -v brass_dvt_allison -V brass_dvt_derek brass.h
```

Derek can then incorporate selected changes with one or more invocations of the *patch* command. If he wants to incorporate all of Allison’s changes, he might prefer to perform a single *merge* command, not a series of *patch* commands.

Note: An invocation of the *patch* command can, in effect, incorporate several versions’ changes into the target version. See the [patch](#) reference page for details.

patchlist merely provides information: the depot-relative pathname and version-ID of each version, along with the number of the *keep* transaction in which that version was created. For example:

```
\\.src\brass.h t:88 (5/23)
\\.src\brass.h t:87 (5/22)
\\.src\brass.h t:86 (5/21)
\\.src\brass.h t:85 (5/20)
```

patchlist does not perform any actual *patch* commands.

Options

-v <from-ver-spec>

Specify the version which has the desired changes. You can use a version-ID (**gizmo_dvt/5**) or a stream (**gizmo_dvt**).

-V <to-ver-spec>

Specify the version that you wish the changes to be incorporated into. You can use a version-ID (**gizmo_dvt/5**) or a stream (**gizmo_dvt**).

patchlist

-t <transaction-1-transaction-2>

A transaction range, used with the **-v** and **-V** options to compare the versions between the two listed transactions.

For the **patchlist** command, a transaction can be represented in the following ways:

- Time keyword: **now**
- Transaction number as a positive integer: e.g. **146** or **23965**
- Transaction number keyword: **highest**

-e <eid>

The element-ID of the element to be analyzed. If you use this option, you cannot also specify an <element-list>.

-fx Display the results in XML format.

Examples

For all elements find the versions in the **gizmo_dvt** stream containing changes that have not yet been incorporated into the **gizmo_test** stream:

```
> accurev patchlist -v gizmo_dvt -V gizmo_test
```

See Also

[merge](#), [mergelist](#), [patch](#)

[Using a Specific Version of an Element](#) on page 14

pop copy files into a workspace or reference tree

Usage

```
accurev pop [ -O ] [ -R ]
    [ -v <ver-spec> -L <location> [ -D ] [ --eol=<eol-type> ] ]
    [ -t <time-spec> ] [ -fx ] { -l <list-file> | <element-list> }
```

Description

The *pop* (“populate”) command loads versions of elements into your workspace. This sounds like a useful and common operation, but in practice, you may never need to use this command.

It is appropriate to use *pop* when you accidentally delete files. There is nothing AccuRev can do if you delete files that you have been editing, but have not yet preserved with *keep*. (But the operating system may have an “undelete” facility, such as the Recycle Bin on the Windows desktop.) If you delete files that you have not modified, but still need (e.g. for searching, building, or testing), you can use *pop* to get “fresh copies” of the files from the depot.

For each element, *pop* retrieves a version from the workspace stream. It’s the version that *should* be in the workspace (but isn’t because you deleted it), or *would* be in the workspace (but isn’t because you hadn’t previously loaded it):

- For a file that you accidentally deleted after the workspace’s most recent update, it is the version that was loaded (or left alone) by that update.
- For a file that you have modified and preserved with *keep*, but then accidentally deleted, it’s the most recently kept version.

Note: If the *pop* command is used with the *-v* option, and any element specified in the *<list-file>* or *<element-list>* has status (**defunct**) in the stream’s default group, the command will fail. If you are using a script to generate a list of files to *pop*, check for this potential error condition.

Overwriting Existing Files

pop is designed to “fill in the blanks”, not to replace existing files. Thus, the *pop* default is to refuse to do anything if it would overwrite any file in the workspace. You can force *pop* to overwrite existing files with the *-O* option. It retrieves the version that was loaded (or left alone) by the most recent update.

Be very careful when using *-O*. If you have modified a file but not yet preserved it with *keep*, then *pop -O* overwrites what might be the only existing copy of the modified file.

Timestamps on Files Copied into the Workspace

By default, each file copied into your workspace by this command has its timestamp set to the current time. If environment variable ACCUREV_USE_MOD_TIME is set, the timestamp is set to the time the version was originally created. (Note: that’s the time the version was created in some user’s workspace stream, not the time it was subsequently promoted to the backing stream.) Setting this environment variable may defeat an optimization in the *stat* command. See [Optimized Search for Modified Files](#) on page 247.

Using “pop” Outside a Workspace

You can use *pop* to copy versions of one or more elements from depot storage to a non-workspace location. Use the *-v* option to specify the version to be copied; use the *-L* option to specify the destination of the

pop

copy operation. Use depot-relative pathnames to specify the elements to be copied. The current directory must not be within a workspace. Use the **-D** option to copy only an element, without recreating its directory structure. Use the **--eol=unix** option or the **--eol=windows** option to specify the type of line terminators to write into the copied versions. (The UNIX/Linux line terminator is NL, and the Windows line terminator is CR-NL.)

To select a specific version of a file for development use inside your workspace, use the **co** command.

Using "pop -t" to access an element from a particular time

Sometimes you want to grab an element from a particular point in time, and do not intend to modify or promote it—for example, to build a software package with a component from last week, to track down a build issue. The **pop -t** command allows you to do this easily, without the necessity of setting up a time-basis stream and then using **pop -v** to access a particular version.

Options

-O Override: suppress the warning message that would appear, and overwrite existing files, even if they have been set to read-only by the operating system.

-R Recurse into subdirectories.

-v *<ver-spec>*

Specify the version to be copied. The **-v** and **-L** options must be used together; and you must specify the elements using depot-relative pathnames.

-L *<location>*

Specify the full pathname of a directory that is not within any AccuRev workspace, where the copies of versions are to be made. The **-v** and **-L** options must be used together; and you must specify the elements using depot-relative pathnames.

-D Pop just the specified element, without building its directory tree. Must be used with **-v** and **-L**.

--eol=<eol-type>

Specify the type of line terminators (**unix** or **windows**) to write into the copied versions. Must be used with **-v** and **-L**.

-t *<time-spec>*

Copy an element based on a point in time or a transaction number. A time-spec can be any of the following:

- Time in *<"YYYY/MM/DD HH:MM:SS">* format, "2007/08/07 20:27:15", for example. Note that you must use quotes when specifying time-spec as *YYYY/MM/DD HH:MM:SS* to ensure that the entire argument is interpreted by the command shell as a single token.
- Time keyword: **now**
- Transaction number as a positive integer: e.g. **146** or **23965**
- Transaction number keyword: **highest**

When using the **-t** option, take into account the possibility of a timewarp. The `AC_SYNC` environment variable (see [AccuRev User Preferences](#) on page 6) determines how a timewarp will be handled. See also section [System Clock Synchronization](#) on page 31 of the *AccuRev Administrator's Guide*.

pop

-l <list-file>

Process the elements listed in <list-file>. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any <element-list> specified is silently ignored.

<element-list>

One or more element names, separated by whitespace. If you also specify a <list-file> using the **-l** option, this <element-list> is silently ignored.

You can include names of directory elements in the list. If you do, be sure to specify the **-R** option.

When using the **-v** and **-L** options, you must specify elements using depot-relative pathnames.

-fx Display the results in XML format. For example:

```
<AcResponse
  Command="pop"
  TaskId="1108">
  <message>Populating element \.\dir00\sub00\file06.txt</message>
  <element location="/dir00/sub00/file06.txt"/>
</AcResponse>
```

Examples

Copy **foo.c** from the workspace stream into the workspace:

```
> accurev pop foo.c
```

Copy versions from the workspace stream into the subdirectory tree starting at the current directory, replacing existing files as well as “filling in the blanks”:

```
> accurev pop -R -O .
```

Using versions from stream **QA**, copy the subdirectory tree starting at the directory **src** to **C:\httpd\test_site**:

```
> accurev pop -R -v QA -L C:\httpd\test_site \.\src
```

Copy only **file.txt** from stream **my_stream** to the current location, without creating the directory structure **dir1/dir2/dir3/dir4**:

```
> pop -L . -D -v my_stream dir1/dir2/dir3/dir4/file.txt
```

Copy only directory **dir4** from stream **my_stream** to the current location, without creating the directory structure above it (**dir1/dir2/dir3**). **dir4** will contain top-level elements. If **dir4** contains subdirectories, they will be created, but empty.

```
> pop -L . -D -v my_stream dir1/dir2/dir3/dir4
```

Copy **file1.txt** from two different directories in stream **my_stream**, to the current location, without creating the directory structures above it. In this case, the second copy (from **dir1**) overwrites the first one (from **dir1/dir2/dir3/dir4**):

```
> pop -L . -D -v my_stream dir1/dir2/dir3/dir4/file1.txt dir1/file1.txt
```

Copy elements associated with transaction id **68456**, overwriting existing files and using recursion:

```
> pop -t 68456 -R -O .
```

pop

Copy elements modified on January 5, 2010 at 1:00, overwriting existing files and using recursion:

```
> pop -t "2010/01/05 01:00:00" -R -O .
```

See Also

update

Using a Specific Version of an Element on page 14

promote propagate a version from one stream to another stream

Usage

```
accurev promote [ -c <comment> ] [ -d | -k | -p ] [ -K ] [ -O ] [ -N ]
  [ -s <from-stream> [ -S <to-stream> ] ] [ -q ]
  [ -I <issue-number(s)> [-3] ]
  { -l <list-file> | <element-list> | -t <transaction-number> | -e <eid>
  | -Fx -l <XML-file> }
```

Description

The *promote* command sends a stream's changes, involving one or more elements, to its parent stream. The most common use of *promote* is to send new versions, created in your workspace stream with the *keep* command, to the backing stream. This has the effect of taking your private changes and making them public.

Other changes that can be promoted include: renaming and moving of elements, creation of new elements, and defuncting/undefuncting of elements.

Promotion-Related Triggers

AccuRev defines these triggers related to the *promote* command:

- A *pre-promote-trig* trigger fires on the client machine before the *promote* operation takes place. You can use this kind of trigger to control promotions, perhaps cancelling the promotion in certain cases.
- A *server-post-promote-trig* trigger fires on the server machine after the *promote* operation takes place. Typically, this kind of trigger notifies one or more users of the promotion.
- The *server_preop_trig* trigger fires on the server machine before the *promote* operation takes place, if the trigger has been configured for the depot in which the elements to be promoted reside.

For more information on triggers, see the [mktrig](#) reference page and [AccuRev Triggers](#) on page 77 of the *AccuRev Administrator's Guide*.

Keep and Promote

Basic development with AccuRev involves editing a file element in your workspace, then using *keep* to create a new version in your workspace stream. This stores a permanent copy of the edited file in the depot. The new version that *keep* creates in your workspace stream is called a real version, because it's directly associated with the new file in the depot.

By contrast, promoting a kept version doesn't store a new file in the depot. Instead, it creates a virtual version in the parent stream. This version is just a reference to (or an alias for) the real version. The virtual and real versions have identical contents: the file originally stored in the depot by the *keep* command. Thus, the effect of *promote* is to make an existing real version available at the next higher level in the stream hierarchy.

Directory Promotion

Directory elements need to be promoted only after they are first *add'*ed to the depot and after they are renamed. AccuRev automatically promotes a directory that doesn't yet exist in the backing stream when you promote any element within that directory.

promote

Promotion and the Default Group

When you promote an element from a “child” workspace or stream to its “parent” stream, the element is removed from the default group of the child stream, and are added to the default group of the parent stream. That is, the elements become inactive in the child stream, and active in the parent stream.

Exceptions:

- Promoting from a time-based stream to its parent does not remove the elements from the default group of the time-based stream.
- Promoting an “old” version of an element (see [Promoting an ‘Old’ Version](#) below) does not remove the element from the default group of the workspace.

Promotions to destinations other than the immediate parent stream do *not* remove elements from the default group of the “child” stream. See [Stream Promotion](#) below.

Promoting an ‘Old’ Version

A common AccuRev scenario involves using *keep* several times to preserve intermediate versions of a file, then using *promote* to make the most recently kept version public. But after you’ve kept (say) five versions, you might decide it’s the third version that deserves to be promoted, not the fifth and final version. There’s no need to “roll back” your workspace to the older version. Instead, you can specify that older version as the version to be promoted.

Note: As always, AccuRev may disallow the promotion of your version because changes made in other workspaces need to be merged into your version. Use the *-O* option to suppress this “is a merge required?” check.

When you promote an older version, the most recent version of the element remains active in your workspace. (That is, it remains in the workspace’s default group.) This enables you to promote a stable, older version of an element, while continuing to work on a newer version that is not yet ready to be promoted.

To promote an older version of an element that is active in your workspace, specify a transaction number with the *-t* option. This must be a *keep* transaction that took place in your workspace, AccuRev promotes all the versions created in that transaction:

```
> accurev promote -t 53
Validating elements.
Promoting elements.
Promoted element \.\src\brass.c
Promoted element \.\src\brass.h
```

Exclusive File Locking and Anchor-Required Workspaces

If exclusive file locking is in effect for an element, *promote* returns it to read-only status. See [File Locking in Workspaces](#) on page 5.

When Promotion Fails

If two or more developers have workspaces that are backed by the same stream, they’re in a first-come-first-served race to promote versions to the backing stream. If somebody else has already promoted his version of any of the files you are promoting, the entire *promote* command is cancelled. For each such file, you must *merge* your colleague’s changes (the changes that he managed to promote before you did) into the version in your workspace. After *keeping* the merged file, you can promote it to the backing stream.

promote

Stream Promotion

By default, *promote* sends changes from your workspace stream to its backing stream. The alternatives are:

- Any non-workspace stream's changes can be promoted to its parent stream. Use the *-s* option to specify the "from" stream.
- Any non-workspace stream's changes can be promoted to a stream that is *not* its parent. Use both the *-s* and *-S* options, to specify the "from" and "to" streams.

Note: In this case, termed a cross-promotion, the promoted versions are *not* removed from the default group of the "from" stream.

If a depot's stream hierarchy is deep, several promotions will be required to propagate an element's changes from their origin (the workspace stream) to their final destination (the depot's base stream).

At any stream level, a merge (*merge -v*) may be required to enable promotion to the next level.

What Promotion Doesn't Do

Promotion affects the workspace stream (located in the depot), but it has no effect on the files in the workspace tree (located in your disk storage). In particular, a promoted element is not removed from your workspace. The file remains in your workspace and is writable. You can immediately edit it without doing any sort of "check-out" or having to run the *co* command.

Integration with AccuWork

Invocation of the *promote* command can activate the change-package-level integration between AccuRev's configuration management and issue management facilities. Before the *promote* operation is performed, you are prompted to enter one or more AccuWork issue record numbers:

```
Validating elements.
```

```
Please enter issue number ?:
```

(Use the *-I* option to specify issue record number(s) on the command line and bypass this prompt.) Enter a number, or multiple numbers separated by SPACES. After the elements are promoted, the change-package-level integration updates the change package (Changes subtab) of the issue record.

For more information, see [The Schema Editor \(Administrators Only\)](#) on page 381 of the *AccuRev On-Line Help*.

Options

-c <comment>

Specify a comment for the transaction. The next command-line argument should be a quoted string. Alternatively, the next argument can be in the form *@<comment-file>*, which uses the contents of text-file *<comment-file>* as the comment.

-d Selects all elements in the default group of the workspace (or more generally, the child stream). When used with the *-t* option, see the *-d -t* explanation below.

-e <eid>

Operates on the element with the specified element-ID. This is useful for promoting stranded elements to the backing stream. Use *stat -i* to list stranded elements. If you also specify a *<list-file>*, *<element-list>*, or *<transaction-number>*, it is ignored.

promote

-I <issue-number(s)>

Bypasses the prompting for an issue record by the change-package-level integration with AccuWork. (See [Integration with AccuWork](#) above.) The integration uses the number you specify after **-I**. You can specify multiple issue records as a SPACE-separated list, enclosed in quotes:

```
-I "349 411 413"
```

Note: This option does not invoke a promote-by-issue operation. Rather, the versions you are promoting are applied to the specified issue record(s). For promote-by-issue, use the **-Fx** option.

-3 Specifies that any <issue-number> specified by the **-I** switch is a third-party ITS key rather than an AccuWork issue number.

-k Selects all kept elements. (Not valid with **-l**, **-s**, or **-p**.)

-K If a specified element has **(modified)** status, first *keep* the element, then *promote* it. If you specify a comment with **-c**, it becomes part of both the *keep* and *promote* transactions. Note that **server_preop_trig** is called twice: once for the *keep* operation, and once for the promote operation.

-N Asserts that the version(s) to be promoted contain all the changes in the change package specified with **-I** (or with the change-package-level integration between AccuRev and AccuWork). AccuRev verifies the “all changes are included” assertion before promoting the version(s); if the assertion is false, the command is cancelled.

No modification is made to any change package; the only change is that an existing change package (the one specified with **-I**) is now “in” an additional stream (the one to which you’re promoting the versions).

Typical use case: you’ve fixed a bug in stream A, and now you wish to incorporate the fix into stream B, as well. You use the *patch* command on one or more elements, sending the changes you made in stream A to a workspace based on stream B. Then, you *promote -N* the new versions to stream B.

Using the **-N** option prevents two unwanted things:

- It suppresses a “change package merge required” error, which would occur when AccuRev attempts to add your new versions (in stream B’s workspace) to the change package (which currently contains versions from another stream).
- It would be wrong to add the new versions to the change package, because the modified change package would no longer be “in” the original stream (stream A).

-p (not valid with **-k** or **-s**) Selects all pending elements. If some of the pending elements have **(modified)** status, use the **-K** option, also.

-q Suppresses the execution of a workflow transition that would otherwise be executed when an issue enters a stream. Applicable only if the stream into which the issue is being promoted has a workflow rule restricting entry to issues in a specific workflow stage *and* a transition has been specified for that stage in the workflow rule. See the *AccuRev Web User Interface User’s Guide* for more information on implementing a workflow.

-s <from-stream>

Promote from the specified stream to its parent stream (or to the stream specified with **-S**). <from-stream> cannot be a workspace stream. See the description of <element-list> below.

promote

-S *<to-stream>*

Promote to the specified stream. You must also specify the “from” stream, with **-s**.

- O** Override: (1) if there’s a name discrepancy, change an element’s name in the “to” stream to match its name in the “from” stream; (2) promote a version even if a *merge* would normally be required with the version in the “to” stream.

Defaults: (1) don’t change an element’s name in the “to” stream (just promote the new version); (2) cancel the entire *promote* transaction if any version requires a merge.

-Fx -l *<XML-file>*

Specifies a set of issue records with an XML-format text file. Example:

```
<issues>
  <id>8</id>
  <id>14</id>
  <id>102</id>
</issues>
```

The versions to be promoted are the head versions in the issues’ change packages (Changes subtab).

-l *<list-file>*

Process the elements listed in *<list-file>*. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in a list-file.

If you use this option, you cannot also specify **-k**, and any *<element-list>* specified is silently ignored.

Please note that using the plain text output of the *mergelist* command as the *<list-file>* when cross-promoting (**defunct**) elements will cause errors. Use the **-Fx** option for the *mergelist* command to include version information in the output, and use the resulting XML file to promote instead.

<element-list>

One or more element names, separated by whitespace. If you also specify a *<list-file>* using the **-l** option, this *<element-list>* is silently ignored.

If you use **-s** to promote from a non-workspace stream, then each name in *<element-list>* is interpreted relative to the top-level directory of the depot. Typically, a simple filename like **base.h** won’t be valid; instead, use a pathname like **src/include/base.h**. Alternatively, use a full depot-relative pathname, like ./src/include/base.h.

-t *<transaction-number | time>*

Promote all the versions created in the specified transaction. See [Promoting an ‘Old’ Version](#) above. If you also specify a *<list-file>* or *<element-list>*, it is ignored. When used with the **-d** option, see **-d -t** explanation below.

-d -t *<transaction-number | time>*

Promote all active transactions up to and including the specified transaction or time. In other words, all active elements versions at the specified transaction id or time. The element versions

promote

that will be promoted will be the same as returned by the "accurev stat -d -t <transaction | time>" command.

Examples

Promote files **commands.c** and **base.h**:

```
> accurev promote commands.c base.h
```

Promote all elements that you have kept, but not yet promoted:

```
> accurev promote -k
```

Promote the head versions in the change packages of the issue records listed in XML file **promote_these_issues.xml** (see the description of **-Fx** above for an example file):

```
> accurev promote -Fx -l promote_these_issues.xml
```

Promote all active elements of a stream <**s**> at or up to transaction <**t**>:

```
> accurev promote -s <s> -d -t <t>
```

See Also

co, *demote*, *keep*, *merge*, *mktrig*

Techniques for Selecting Elements on page 10

purge undo all of a workspace's changes to an element

Usage

```
purge [ -c <comment> ] [ -s <stream> ] [ -I <issue-number(s)> [ -3 ] ]
{ <element-list> | [ -Fx ] -l <list-file> | [ --rebase] [ -i ] | -e <eid> }
```

Description

The *purge* command undoes all of the changes you have made to an element since you activated it in your workspace — that is, since you added the element to the workspace's default group. Thus, *purge* cancels changes that you have not yet promoted to the backing stream:

- It removes the element from your default group (deactivates the element in your workspace stream).
- It replaces the file in your workspace with a copy of the version that was in the backing stream at the time of your most recent *update*. (But if you promoted one or more versions of the element to the backing stream since your most recent *update*, it restores the most recently promoted version to your workspace.)

purge does not actually remove any version of the element. (Nothing does that — AccuRev is TimeSafe.) You can still access versions that you created with *keep*, but then *purged*, with such commands as *hist*, *co*, and *pop*.

To remove an element from your workspace, use the *defunct* command.

Timestamp on the Restored Version

By default, each file copied into your workspace by this command has its timestamp set to the current time. If environment variable ACCUREV_USE_MOD_TIME is set, the timestamp is set to the time the version was originally created. (Note: that's the time the version was created in some user's workspace stream, not the time it was subsequently promoted to the backing stream.) Setting this environment variable may defeat an optimization in the *stat* command. See [Optimized Search for Modified Files](#) on page 247.

Effect of “purge” on “defunct”, “undefunct”, and “move”

purge undoes all changes that you have not yet promoted, including removal of elements with *defunct* and renaming/moving of elements with *move*. For example, if you defuncted or moved element **foo.c** (but didn't *promote* the change), a subsequent *purge* makes **foo.c** reappear in your workspace. If you have defuncted a directory, thus removing all of its contents from your workspace, purging the directory will bring all of the contents back.

Similarly, if you purge an element that you've resurrected with *undefunct*, the element disappears again!

To undo a purge

AccuRev does not have a command to revert a *purge* operation, but the following workaround allows you to restore purged elements through the GUI. In the StreamBrowser:

1. Create a new stream off the stream where the *purge* was performed and set its time stamp to 1 second before the *purge* operation.
2. Right-click the new stream and choose **Send to Change Palette**. The pointer changes to include the Change Palette icon.

purge

3. Select the stream where the purge took place. The Change Palette appears.
4. In the Change Palette, select the elements that were purged and promote them back to the stream where the *purge* was performed.

Exclusive File Locking and Anchor-Required Workspaces

If exclusive file locking is in effect for an element, *purge* returns it to read-only status. See [File Locking in Workspaces](#) on page 5.

Options

-c <comment>

Specify a comment for the transaction. The next command-line argument should be a quoted string. Alternatively, the next argument can be in the form @<comment-file>, which uses the contents of text-file <comment-file> as the comment.

-s <stream>

Stream to purge changes from. Use this option with extreme caution. Like *accurev promote -s*, it removes pending (not yet promoted) changes from the specified stream. See the description of <element-list> below.

Note: invoking *purge -s* causes the depot's **pre-promote-trig** trigger to fire, because it changes which version of the purged element appears in the specified stream.

-I <issue-number(s)>

Bypasses the prompting for an issue record by the change package integration with AccuWork. The integration uses the number you specify after **-I**. You can specify multiple issue records as a SPACE-separated list, enclosed in quotes:

```
-I "349 411 413"
```

-3

Specifies that any <issue-number> specified by the **-I** switch is a third-party ITS key rather than an AccuWork issue number.

-Fx Signals that <list-file> (see the **-l** option) is an XML-format file, not a flat text file. Example:

```
<elements>
  <e eid="2"></e>
  <e eid="200"></e>
</elements>
```

-l <list-file>

Process the elements listed in <list-file>. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any <element-list> specified is silently ignored.

<element-list>

One or more element names, separated by whitespace. If you also specify a <list-file> using the **-l** option, this <element-list> is silently ignored.

purge

If you use `-s` to purge changes from a non-workspace stream, then you must specify each name in `<element-list>` with a depot-relative pathname or an element-ID (`-e` option).

`--rebase`

Purge elements with changes that are already in the backing stream. You can use this option instead of specifying elements. When you perform a rebase operation on a stream or workspace, AccuRev purges elements if the current version is the same as what is in the backing stream or if the element is in an underlapped state. If you use the `-i` option with `--rebase`, AccuRev also purges stranded elements.

Note: You can also perform a rebase operation by running the `rebase` command.

`-i` Remove stranded elements from the default group. If you want to identify any stranded elements that might be purged by the rebase operation, you can run `stat -i` prior to running `--rebase`.

`-e <eid>`

Operate on the element with the specified element-ID. You can use this option instead of specifying a list of elements. This option cannot be used in combination with the `--rebase` option, the `-i` option, the `-l` option, or an element list.

Examples

Purge changes you've made to files `commands.c` and `base.h`:

```
> accurev purge commands.c base.h
```

Promote a version of file `tools.readme` to stream `tulip_test`, then “undo” it:

```
> accurev promote tools.readme
Validating elements.
Promoting elements.
Promoted element \.\tools\tools.readme
```

```
> accurev purge -s tulip_test \.\tools\tools.readme
Purging element \.\tools\tools.readme
```

See Also

`co`, `defunct`, `hist`, `rebase`, `undefunct`

Techniques for Selecting Elements on page 10

putconfig update the contents of an AccuWork configuration file

Usage

```
accurev putconfig [ -p <depot-name> ] [ -u <user-name> ]
                 -r <config-file> -l <new-config-data>
```

Description

The *putconfig* command replaces the contents of an AccuWork configuration file with the file you specify on the command line.

WARNING: before using this command, be sure you have a good backup of subdirectory **dispatch/config** of the depot's storage directory.

See the [getconfig](#) reference page for a listing of the configuration file names and their contents.

Options

-p <depot-name>

The depot to use (default: the depot of the current workspace).

-u <user-name>

The AccuRev user whose configuration file is to be updated. Use this option when updating a user's personal AccuWork queries (*-r query.xml*).

-r <config-file>

The name of the XML-format configuration file whose contents are to be updated.

-l <new-config-data>

The name of the XML-format file containing the configuration data to be stored in <config-file>. The best way to create this file is to edit the output of a *getconfig* command.

Examples

Use the contents of file **c:\temp\q.new** to establish a new set of AccuWork queries for user **derek**, in the current depot's issues:

```
accurev putconfig -u derek -r query.xml -l c:\temp\q.new
```

See Also

[getconfig](#)

reactivate

restore a reference tree, stream, user, group, or workspace to active service

Usage

```
accurev reactivate ref <reftree>
accurev reactivate stream <stream>
accurev reactivate user <principal-name>
accurev reactivate group <group-name>
accurev reactivate wspace <workspace>
```

Description

The *reactivate* command makes a previously *removed* reference tree, stream, user, group, or workspace available for use again.

You cannot use *remove* on elements. The comparable commands for elements are *defunct* and *undefunct*.

Examples

Make stream **gizmo** accessible again, after having previously deactivated it:

```
> accurev reactivate stream gizmo
```

See Also

defunct, *remove*, *show*, *undefunct*

rebase

purge elements with changes that are already in the backing stream

Usage

```
accurev rebase [ -c <comment> ] [ -s <stream> ] [ -i ]
```

Description

The **rebase** command purges elements in your default group according to specific criteria. When you perform a rebase operation, AccuRev purges elements from a stream or workspace when the current version of the element is:

- the same as what is in the backing stream
- underlapped with the backing stream
- stranded

A rebase operation is useful when you want to allow inheritance, resolve underlaps, or purge stranded elements.

In some situations, such as when you change a stream basis time from some time in the past to something more current, there may be files in your default group for which inheritance is blocked. You can use the **rebase** command as a quick way to purge these files so that inheritance can take place.

Another scenario in which a rebase operation is useful is after you have reparented a stream in such a way as to create underlapped files. Rebasing the stream purges the underlapped files and resolves the underlapped state.

Note: Rebase is also an option of the **purge** command.

Options

-c <comment>

Specify a comment for the transaction. The next command-line argument should be a quoted string. Alternatively, the next argument can be in the form **@<comment-file>**, which uses the contents of text-file **<comment-file>** as the comment.

-s <stream>

Stream to purge changes from. If you do not specify a stream, AccuRev rebases the workspace associated with your current directory location.

Note: invoking **rebase -s** causes the depot's **pre-promote-trig** trigger to fire.

-i

Purge any stranded elements. Stranded elements are members of the default group that no longer have a pathname in the workspace or stream.

Examples

Rebase stream **gizmo**:

```
> accurev rebase -s gizmo
```

rebase

See Also

purge

reclaim

reclaim remove archived version container files from gateway area

Usage

```
accurev reclaim [ -p <depot> ] -t <archive-transaction>
```

Description

The **reclaim** command removes archived version container files from a depot's gateway area. You must specify the transaction number of a previous **archive** command. The container files of all the versions processed by that archive command are removed from the depot's gateway area.

For a complete description of archiving, see [Archiving of Version Container Files](#) on page 37 of the *AccuRev Administrator's Guide*.

See Also

[archive](#), [unarchive](#)

remove

remove

remove user sessions or deactivate AccuRev objects

Usage

```
accurev remove group <group-name>
accurev remove ref <reftree>
accurev remove sessions <user-name>
accurev remove stream <stream>
accurev remove user <principal-name>
```

Description

The **remove** command hides a reference tree, stream, user, group, or workspace. It does not remove the object from the depot — that would violate AccuRev’s TimeSafe property. It omits the object from **show** command listings — but the **show -fi** command does display such “invisible” objects.

Many operations on “removed” objects are disabled. For example, you cannot work in a removed workspace or update a removed reference tree, and you cannot perform work as a removed user. But you can promote versions to a removed stream.

If you make a mistake creating an object, you may want to **remove** it. But since this command does not actually delete the object from the depot, you cannot create a new object with the same name.

Use the **reactivate** command to undo the removal of an object.

Removing User Sessions

The special command form **remove sessions** terminates all login sessions for the named user. Unlike other items which can be removed with the **accurev remove** command, once the login sessions are removed, they cannot be reactivated. The user whose sessions have been removed can log in again as needed, as long as they are otherwise allowed to by the system. This command activates the **server_admin_trig** trigger, which can be used to place restrictions on who can execute it.

Note: In a replicated environment, you must run the **accurev remove sessions** command from the AccuRev master, even if the user you whose session you want to end logged in to a replica server.

Deactivating a User’s Workspace

Use the **rmws** command to deactivate (hide) one of your workspaces. To deactivate another user’s workspace, use the **rmws -s** command.

Examples

Hide stream **gizmo**:

```
> accurev remove stream gizmo
```

Deactivate user **tom**:

```
> accurev remove user tom
```

See Also

[reactivate](#), [rmws](#), [show](#)

replica

perform operations on a replica repository

Usage

```
accurev replica sync
```

```
accurev replica -t <transaction-range> [ -i ] [ -fx ] [ -p <depot-name>
archive
```

Description

replica sync

The *replica sync* command brings your local replica repository up to date with respect to the master repository.

During the course of development, your local replica repository typically becomes out-of-date with respect to the master repository. This occurs when other users send commands to other replica servers or directly to the master server. In both such cases, new transactions are entered in the master repository, but are not entered in the your local replica repository.

At any time, you can enter a *replica sync* command to bring your local replica repository up to date. This transfers data from the master repository site slice to the replica repository site slice. It also transfers database transactions from the master repository to the replica repository — but only for the depots that are included in the local replica. The *replica sync* command does not transfer the corresponding storage files for *keep* transactions.

A *replica sync* command is performed automatically on the local replica after each operation that is initiated by a client of the local replica, and that makes a change to the repository.

Note: You never need to synchronize directly with other replicas; synchronizing with the master is sufficient to bring your replica up to date.

replica archive

The *replica archive* command is used to remove unused container files from a depot's **storage** directory on a replica server. This command activates the **server_preop_trig** trigger on the replica with a *replica_archive* command.

The command output lists each element for which the storage container was removed, for example:

```
Removing storage for element /file1.txt (2/5)
Removing storage for element /file1.txt (2/6)
```

Options

These options apply to the *replica archive* command only.

-t <transaction-range>

Specifies a transaction range to the replica archive command. The command looks for **archive** transactions in the range and physically removes the container files of all the versions on the replica associated with those transactions. Transactions can be specified in the following ways:

- Time in <YYYY/MM/DD HH:MM:SS> format: e.g. **2007/08/07 20:27:15**.
Note that you might need to use quotes in composing the argument following -t; the entire argument must be interpreted by the command shell as a single token.
- Time keyword: **now**
- Transaction number as a positive integer: e.g. **146** or **23965**
- Transaction number keyword: **highest**

-i Lists the versions that would be archived (in XML format), but does not actually archive them.

-fx Display the results in XML format.

-p <depot-name>

Specifies the depot to use; defaulting to the depot of the current workspace. If this option is not specified, you must be in a workspace.

See Also

archive, mkreplica, rmreplica

[Replication of the AccuRev Repository](#) in the *AccuRev Administrator's Guide*

revert

“undo” a promote transaction

Usage

```
accurev revert [ -I <issue-number> [ -3 ] ] [ -G ] [ -K ]
               [ -S <stream> ]

accurev revert -t <transaction-number> [ -G ] [ -K ] [ -S <stream> ]
]
```

Description

The **revert** command provides an “undo change” capability for dynamic streams. You have the choice of specifying a “traditional” revert operation, which requires you to be in a workspace based on that stream, or a “direct” revert operation, which does not require a workspace.

The “direct” **revert** takes fewer steps, but since it places the results of the **revert** directly into a stream, it is best used in simpler scenarios, or in a stream that does not propagate immediately to other users. For more complicated scenarios, where you want to test the results of the **revert**, it is better to use the workspace-based “traditional” **revert**.

revert doesn't actually remove any versions from the stream or transactions from the repository; that would be a violation of AccuRev's TimeSafe principle. Instead, **revert** creates a new version (using **keep**) of one or more elements. For a traditional **revert**, this **keep** is performed in your workspace; for a direct **revert**, the **keep** is done within the stream. If the version being reverted is the first version in a stream, the command **defuncts** that element.

Note: You cannot direct revert an initial version, unless there are no other changes in the backing stream. This means that you cannot revert the first version of a file if someone else has already made changes to it. You cannot defunct a file that someone else edited because it would not be possible to then add their edits back into the non-existent file.

The contents of the new version are the result of “subtracting out” a certain set of changes from the stream's current version:

- the changes that were added to the stream in a specified promote transaction, or
- the changes contained in a change package that is currently in the stream.

The “subtracting out” of content changes is performed by the same tool that performs a content-level **merge**. Submitting a different set of versions to this tool effectively implements a “reverse patch” algorithm (see below).

The **revert** command also “subtracts out” namespace changes. If the file name and/or parent directory were changed by the issue/transaction being reverted, and no one has moved or renamed the file since this issue/transaction, AccuRev moves the file back to its pre-issue/transaction directory and give it back its old name.

After a traditional **revert** command completes, you can verify the correctness of its work (and make further changes, if appropriate) in the workspace. Then, you can complete the “undo change to stream” process by promoting the new versions.

A direct **revert** promotes its changes directly into the specified stream. However, we recommend that you still build and test these changes in an updated workspace to verify the results.

revert

Exclusive File Locking and Anchor-Required Workspaces

See [File Locking in Workspaces](#) on page 5.

Reverse Patch Algorithm

In a **revert** command, the changes in a specified set of versions are removed from a stream's current versions of those elements:

- When reverting a transaction, AccuRev regards each version in that transaction as being the head version of a patch, and removes all the changes in that patch from the element. The basis of the patch is the version of the element that was visible in the stream at the transaction before the one being reverted (that is, “revert transaction - 1”).
- When reverting a change package (specified by the number of the issue record containing the change package), AccuRev removes all the changes in each element's change package entry.

For each element it processes, the **revert** command uses the same tool as the **merge** command to perform the operation that removes a set of content changes from the current version. We use the term “reverse patch” to describe this process, but it's really just another instance of effectively modifying the merge algorithm by switching around the versions.

When the merge tool is invoked by **revert**:

- The version you selected when invoking the **revert** command is designated to be the closest common ancestor.
- The version currently in the stream is designated to be the “from version”.
- For revert by transaction, AccuRev determines the version that was in the stream just before the current version was promoted there; this version is designated to be the “to version”. For revert by issue, the common ancestor is the issue basis, and the “to version” is the issue head.

Just as with the **merge** command, you can use environment variable AC_MERGE_CLI to configure an alternative merge tool. See [User Preferences](#) on page 156.

revert finishes its work by invoking the **keep** command to preserve the results of the merge (reverse patch) operation.

In this algorithm's switching around of the versions, the basis version of the change package entry becomes the direct ancestor of the newly created version. This listing of the final keep transaction shows the result of removing a change package entry whose head version is 9/14 and whose basis version is 9/17.

```
transaction 214; keep; 2006/07/18 13:44:41 ; user: mary
# Revert issue # 12
\.\dir04\sub04\file07.txt 8/1 (8/1)
ancestor: (9/14) reverted from: (9/23)-(9/17)
```

The “reverted from” data indicates that the changes made between versions 9/17 and 9/23 were *added* to the common ancestor version, 9/14. This is equivalent to *subtracting* the changes in the change package entry (9/14 - 9/17) from the current version, 9/23.

To revert a purge operation

AccuRev does not have a command to revert a **purge** operation. However, you can use the procedure documented at the **purge** command. In the *CLI User's Guide*, see [To undo a purge](#) on page 204.

Options

-t <*transaction-number*>

Select all elements involved in the specified *promote* transaction. (No other kind of transaction can be reverted.) You must specify the transaction by number. Either **-t** or **-I** must be specified, but not both.

-I <*issue-number*>

For a typical *revert* operation, **-I** <*issue-number*> specifies the issue record whose change package is to be reverted. When performing a “traditional” *revert*, which requires the use of a workspace, you must specify either **-t** or **-I**, but not both. When performing a direct *revert*, you can specify the full **-I** (specifying both source and destination issue numbers) with **-S**.

-3

Specifies that the <*issue-number*> specified by the **-I** switch is a third-party ITS key rather than an AccuWork issue number.

-G Use the graphical Merge tool to perform the merge if there are conflicts (which require human interaction).

-K If the reverse patch operation does not involve any conflicts (and thus, no human intervention is required), perform this operation and *keep* the new version without asking for confirmation.

-S Perform a direct revert of a change package in the specified stream, without using a workspace. This argument requires either an issue (**-I**) or transaction (**-t**) specification.

Examples

“Undo” *promote* transaction 489, by subtracting the recent changes in the versions promoted by that transaction.

```
> accurev revert -t 489
```

“Undo” the changes recorded in the change package of issue record #4519, bypassing reverse-patch confirmations when there are no conflicts:

```
> accurev revert -K -I 4519
```

Use a “direct” *revert* operation to undo the changes recorded in the change package of issue record #2763 in stream prod1_0_itr:

```
> accurev revert -S prod1_0_itr -I 2763
```

Use a “direct” *revert* operation to undo the changes recorded in the change package of issue record #15 in stream prod1_0_itr:

```
> accurev revert -S prod1_0_itr -I 15
```

See Also

anc, *anchor*, *co*, *defunct*

rmmember

rmmember

remove a user from a group

Usage

```
accurev rmmember <principal-name> <group-name>
```

Description

The *rmmember* command removes an AccuRev user from an AccuRev group.

Examples

Remove AccuRev user **john_smith** from AccuRev group **eng**:

```
> accurev rmmember john_smith eng
```

See Also

addmember, *show* (groups, and members)

rmproperty remove property from stream or principal

Usage

```
accurev rmproperty [ -s <stream> | -u <principal> ] <name> [ -r ]
```

Description

The *rmproperty* command removes a property from a stream or a principal (user or group).

If you want to restrict the ability to remove properties from a stream or principal, you need to modify the `server_admin_trig` server-side trigger. See [Format of the “server_admin_trig” Trigger Parameters File](#) on page 94 of the *AccuRev Administrator’s Guide*.

Options

-s <stream>

Specifies the name of the stream from which to remove the property.

-u <principal>

Specifies the name of the user or group from which to remove the property.

<name>

Specifies the name of the property to remove.

-r

Option required to remove a reserved property. Required to remove the `streamStyle` property, for example.

Examples

Remove the property named "Status" from stream "prod_2010":

```
> accurev rmproperty -s prod_2010 Status
```

Remove the property named "streamStyle" from stream "nightly":

```
> accurev rmproperty -s nightly streamStyle -r
```

Remove the property named "contractor" from user "jsmith":

```
> accurev rmproperty -u jsmith contractor AcmeTemps
```

Modifications to server_admin_trig

If you have implemented the server-side trigger `server_admin_trig` at your site, you need to modify it if you want it to work with the `rmproperty` command. See [The Trigger Parameters File](#) on page 82 in the *AccuRev Administrator’s Guide* for more information.

See Also

[setproperty](#), [getproperty](#), [show](#)

rmreplica remove a depot from a replica repository

Usage

```
accurev rmreplica -p <depot-name>
```

Description

The *rmreplica* command removes the specified depot from the local replica repository's set of depots. The depot becomes inaccessible to clients of the local AccuRev Server. This command must be directed to an AccuRev Server process that manages a replica repository; the corresponding master repository must contain the specified depot.

rmreplica changes the site slice (typically, directory `.../storage/site_slice`) in both the master and replica repositories. Information about which depots are replicated at a site is stored in file `replica_site.xml`, in the local replica repository's site slice. *rmreplica* does not delete the depot's entry from this file; it merely marks the depot as "inactive".

rmreplica does not remove the depot's slice — that is, the depot's storage directory — which is typically located under `.../storage/depots`.

Options

-p <depot-name>

The depot to be removed from the local replica repository.

Examples

Remove depot **kestrel** from the local replica repository.

```
> accurev rmreplica -p kestrel
```

See Also

mkreplica, *replica* (sync)

rmtrig

rmtrig

deactivate a trigger in a depot

Usage

```
accurev rmtrig [ -p <depot-name> ] [ pre-create-trig | pre-keep-trig |  
pre-promote-trig | server-post-promote-trig ]
```

Description

The *rmtrig* command removes the specified trigger.

Options

-p <depot-name>

Specify the depot in which the trigger is to be deactivated. By default, it's the depot for the current workspace.

Examples

Remove the trigger that fires before each *promote* transaction in depot **gizmo**:

```
> accurev rmtrig -p gizmo pre-promote-trig
```

See Also

mktrig, *show* (triggers)

rmws

deactivate a workspace

Usage

```
accurev rmws [ -s ] <workspace-name>
```

Description

The *rmws* command makes the specified workspace inactive. More precisely, it deactivates the workspace stream in the depot; the workspace tree on the user's disk remains unaffected.

The deactivated workspace cannot be used for active development. It won't appear in *show wspaces* listings, unless you use the *-fi* or *-fl* option.

Note: After performing an *rmws* command, you cannot then create a new workspace with the same name. The name remains irrevocably associated with the inactive workspace.

You can use the *reactivate* command to bring the workspace back into active service.

Options

-s Use this option when you deactivate a workspace that belongs to another user. In this case, you must include the *<principal-name>* suffix, in order to fully specify the workspace name.

If you omit this option and specify a *<workspace_name>* with no suffix, *rmws* implicitly adds a *<principal-name>* suffix, using your own principal-name.

Examples

Remove your own workspace, named *talon_dvt_<your-principal-name>*:

```
> accurev rmws talon_dvt
```

Remove another user's workspace, specifying its complete name:

```
> accurev rmws -s talon_dvt_mary
```

See Also

reactivate, *remove*, *show* (wspaces)

secinfo

show user's authorization level

Usage

```
accurev secinfo
```

Description

The *secinfo* (“security information”) command displays a keyword that represents your AccuRev authorization level.

‘AccuRev login’ authorization method

With this authorization method, you establish your AccuRev user identity with the *login* command.

- **authuser** — you are logged in with a username that has a non-empty password.
- **anyuser** — you are logged in with a username that has an empty password.
- **notauth** — you are not logged in.

‘Custom’ authorization method

With this authorization method, you establish your AccuRev user identity (“principal-name”) using some external user-authentication method. For example, to authenticate via an LDAP server, an LDAP authentication trigger intercepts the normal processing of the *login* command, and uses the specified login name to authenticate against the login names stored on the LDAP server. Each username used to access AccuRev must exist both in the AccuRev's user database and on the LDAP server for the authentication trigger to work. The AccuRev user name must match the user on the LDAP server exactly.

For more information on user authentication and access control in AccuRev, see:

- [AccuRev Security Overview](#) on page 57 in the *AccuRev Administrator's Guide*
- The topics under “AccuRev Security” in the AccuRev GUI online help

Examples

Display your AccuRev authorization level, before and after logging in.

```
> accurev secinfo
notauth
```

```
> accurev login john
Password: *****
```

```
> accurev secinfo
authuser
```

See Also

login, logout

[AccuRev Security Overview](#) on page 57 in the *AccuRev Administrator's Guide*

setacl

set or remove an access control list entry

Usage

```
accurev setacl { depot <depot-name> | stream <stream-name> }
  { anyuser | authuser | <user-name> | <group-name> }
  { none | all | clear } [ inherit | noinherit ]

accurev setacl [ depot <depot-name> | stream <stream-name> ] clearall
```

Description

The *setacl* command changes the access control list (ACL) for a depot or stream. (For controlling ACLs on individual elements, see the *eacl* command.) For this command, “stream” includes dynamic streams, workspace streams, and snapshots. This command either creates an ACL entry (also called a permission), or deletes one or more ACL entries.

Each permission controls the rights of one or more users to access the data within the specified depot or stream. Permissions on a depot can be inherited by all of its streams; permissions on a stream can be inherited by lower-level streams.

By default, AccuRev is wide open: all users can access all depots and all streams within the depots.

ACL Permissions and Time Considerations

ACL permissions apply to a stream regardless of any basis time on the stream. Similarly, ACL permissions can be placed on a snapshot, even though such permissions are necessarily created after the snapshot is created.

Commands that Check Permissions

The following commands check ACL entries on one or more workspaces/streams before proceeding. For instance, if a user does not have access to the **gizmo** stream, then the command *accurev cat -v gizmo myfile.c* causes a not-authorized error.

If a version is being accessed from stream A, and that version is cross-linked to stream B, AccuRev checks the ACL permissions on stream A only, not on stream B.

In the following, “stream” can be a workspace stream, dynamic stream, or snapshot.

- *anchor*, *defunct*, *files*, *pop*, *purge*, *stat*, and *update* check the current workspace.
- *annotate*, *cat*, and *co* check the stream of the version being accessed. *co* also checks the current workspace.
- *promote* checks the stream to which the version(s) are being promoted.
- *incl -b* and *clear* check both streams involved in the cross-link.
- *diff* checks the streams of both versions being compared.
- *merge* checks the workspaces/streams of both contributor versions, but not the closest common ancestor version.
- *rmws*, *rmstream*, *reactivate wspace*, and *reactivate stream* check the stream being changed.

setacl

- *mkws*, *mkstream*, and *mksnap* check the specified backing stream for the workspace/stream/snapshot being created.
- *chws* and *chstream* check the stream being changed (and, if appropriate, its new backing stream).
- *show streams* checks the depot.
- *hist -s* checks the stream.

Setting ACL Permissions

setacl commands that create permissions all follow the same pattern:

- Specify the data structure:
 - *depot* <depot-name> sets a permission that controls access to all the data within a particular depot. This includes the AccuWork issues, if any, stored in the depot.
 - *stream* <stream-name> sets a permission that controls access to all the data within a particular stream hierarchy in a particular depot. (There is no need to specify the depot, because stream names are unique throughout the repository — i.e. across all depots.)

A permission on either kind of data structure can be inheritable (see below). An inherited permission created at a given level can be overridden at a lower level.

- Specify the user or set of users:
 - *anyuser* specifies all users who *do not* have a password.
 - *authuser* specifies all users who *do* have a password.
 - <user-name> specifies a particular AccuRev user.
 - <group-name> specifies all users in a particular AccuRev group.
- Specify the access level to be granted:
 - *all* grants access to the data in the specified data structure to the specified users.
 - *none* prohibits access to the data in the specified data structure for the specified users.

Section [Commands that Check Permissions](#) above details the meaning of “access”.

- (optional) Specify a flag that specifies the inheritability of the permission:
 - Depot permission (default = **inherit**): **noinherit** specifies that the permission will apply only to the AccuWork issues in the depot. **inherit** specifies that the permission will also apply to all of the depot’s streams.
 - Stream permission (default = **noinherit**): **noinherit** specifies that the permission will apply only to the specified stream. **inherit** specifies that the permission will also apply to all streams below it.

Note: For a user with an AccuWork-only license, a depot-level determines whether that user can access the AccuWork issues in that depot. Such a user cannot access any of the depot’s version-controlled files, regardless of the stream-level permissions.

Conversion of Pre-Existing Permissions

Prior to AccuRev 4.5, permissions were not inheritable. When a pre-4.5 repository is upgraded to AccuRev 4.5 or later:

- Each existing depot permission is assigned the **inherit** flag.
- Each existing stream permission is assigned the **noinherit** flag.

setacl

Multiple and Conflicting Permissions

Any number of permissions can apply to the same depot or stream. For example, to grant three users access to stream **kestrel_tst**:

```
accurev setacl stream kestrel_tst tom all
accurev setacl stream kestrel_tst dick all
accurev setacl stream kestrel_tst harry all
```

Two or more permissions on a resource can apply to the same user, or to the same depot or stream. In such cases, an **all** permission overrides one or more **none** permissions. This makes it easy to implement “all but” access controls. For example, these permissions prevent everyone in the **famgrp** group — except for users **justine** and **mary** — from accessing stream **spider_dvt**:

```
accurev setacl stream spider_dvt famgrp none
accurev setacl stream spider_dvt justine all
accurev setacl stream spider_dvt mary all
```

An explicit permission on a lower-level stream overrides an inherited permission.

Removing ACLs

To delete an individual permission, use the **clear** keyword:

```
accurev setacl stream spider_dvt mary clear
```

To delete all the permissions for a particular depot or stream, use the **clearall** keyword:

```
accurev setacl stream spider_dvt clearall
```

To delete all the permissions for the entire repository, use the **clearall** keyword without specifying a depot or stream (use with caution!):

```
accurev setacl clearall
```

Examples

Grant access to depot **gizmo** only to users who have passwords:

```
> accurev setacl depot gizmo anyuser none
> accurev setacl depot gizmo authuser all
```

Grant access to stream **talon_tst** only to user **andy**:

```
> accurev setacl stream talon_tst anyuser none
> accurev setacl stream talon_tst authuser none
> accurev setacl stream talon_tst andy all
```

Grant access to the entire stream hierarchy below **gizmo_mnt** to user **mary**:

```
> accurev setacl stream gizmo_mnt mary all inherit
```

Remove all ACL permissions on stream **talon_tst**:

```
> accurev setacl stream talon_tst clearall
```

See Also

chpasswd, eacl, lsacl, mkuser

setpref set one or more user preferences

Usage

```
accurev setpref <pref-name> <pref-value>
accurev setpref -l <XML-file>
```

Description

The *setpref* command places (or replaces) one or more entries in your XML-format AccuRev user preferences file: **preferences.xml** in the **.accurev** subdirectory of your AccuRev home directory.

(By default, the AccuRev home directory is determined automatically; but you can specify an AccuRev home directory with environment variable ACCUREV_HOME.)

You can specify a single name/value pair on the command line. Alternatively, you can specify any number of name/value settings in an XML-format file. The simplest way to create such a file is to modify the output of a *getpref* command.

Note: The name that you specify does not need to be an AccuRev-recognized keyword. For example, this works:

```
accurev setpref foo BAR
... creating this entry in the preferences file:
<foo>BAR</foo>
```

Users Preferences and Environment Variables

If an environment variable has the same name as an entry in the user preferences file, AccuRev commands follow the environment variable and ignore the preference-file entry.

Options

-l <XML-file>

Specifies the location of an XML-format file containing preference settings.

Examples

Set an individual preference:

```
accurev setpref ACCUREV_USE_MOD_TIME 1
```

Set several preferences using an XML-format file:

```
> type mysettings.xml
<AcRequest>
  <AC_DIFF_GUI>TkDiff</AC_DIFF_GUI>
  <AC_MERGE_GUI>Guiffy</AC_MERGE_GUI>
</AcRequest>
> accurev setpref -l mysettings.xml
```

`setpref`

See Also

[getpref](#)

[AccuRev User Preferences](#) on page 6

setproperty set property on stream or principal

Usage

```
accurev setproperty [ -s <stream> | -u <principal> ] [ -r ]
[ <property-name> <value> | -l <file-name> <property-name> ]

accurev setproperty -s <stream> -r [ streamStyle | streamCustomIcon |
stagingStreamResult ] "<xml>"

accurev setproperty -c -r userSchema -l customUserProps.xml
```

Description

The *setproperty* command assigns a property name/value pair to either a stream or a principal (user or group). If the property does not exist, AccuRev creates it; that is, a new property does not require a separate create command. Properties are not TimeSafe; if you respecify or remove a property, you cannot roll back to the prior property name/value pair.

Use *setproperty* command to create custom user properties using the *customUserProps.xml* file. The custom properties display in the user dialogs. Note that the *customUserProps.xml* file can have any name or path you give it.

If you want to restrict the ability to set properties on a stream or principal, you need to modify the **server_admin_trig** server-side trigger. See [Format of the “server_admin_trig” Trigger Parameters File](#) on page 94 of the *AccuRev Administrator’s Guide*.

Options

-s <stream>

Specifies the name of the stream on which to set the property.

-u <principal>

Specifies the name of the user or group on which to set the property.

<property-name>

Specifies the name of the property. Maximum of 255 characters. Property names are case-insensitive regardless of depot settings or platform. The names *title* and *Title* are equivalent, for example.

Tip: You can set the background color of streams and workspaces using the **streamStyle** reserved property. See [Setting Stream Color](#) for more information.

<value>

Specifies the property value. AccuRev places no limit on value length, but length limits may be established by the operating system shell for values input on the command line.

-l <file-name>

To specify a file as a value (an image, for example), use **-l** with the **<file-name>** option. You can use either a relative or absolute path to specify the file location. If you do not specify a path, AccuRev assumes the file is in the current directory. Note that when you use **-l**, you must reverse the order of the name/value pair to value/name.

setproperty

The **-l** option transfers the file content to the AccuRev Server. Similarly, if a file name is specified when *reading* a stream or principal property (using **getproperty**, for example), AccuRev places the content in that file; otherwise, AccuRev returns the file name for that property. See [getproperty](#) for more information.

"**<xml>**"

XML input, enclosed with quotation marks, used to specify the value for the **streamStyle** (for color) and **streamCustomIcon** (for displaying icons and tooltips) reserved properties. See [Setting Stream Color](#) and [Setting Stream Icons](#) for more information. You can use both reserved properties for a stream, but each must be specified separately.

-r

Option required to designate a property as reserved. Required for **streamStyle**, **streamCustomIcon**, **stagingStreamResult**, and custom user properties. Reserved properties are not displayed in the StreamBrowser's Property dialog box.

-c

Designates configuration.

Setting Stream Color

You can use the reserved property **streamStyle** to set the background color of a stream or workspace. The **streamStyle** property takes XML in the following format as its argument:

```
"<style>
  <color>
    <background-color>hex_value</background-color>
  </color>
</style>"
```

The preceding example is formatted to make it easy to read, but you should enter it as a single string when you specify the value for the **streamStyle** property. For example:

```
"<style><color><background-color>#00CC33</background-color></color></style>"
```

Note that the XML must be enclosed in quotes, and that the color (the **<background-color>** tag) is expressed as a hex value (**#FF0000**, for example). AccuRev supports web-safe colors.

Setting Stream Icons

You can use the reserved property **streamCustomIcon** to display an icon on a stream. AccuRev allows you to set:

- Icons that indicate running, success, warning, and failed statuses
- Tooltips that appear when the pointer is placed over the icon
- A URL that opens the default browser at the specified address when clicked

The **streamCustomIcon** property takes XML in the following format as its argument:

```
"<streamicon>
  <image>running | success | warning | failed</image>
  <clickurl>desired URL, http://example.com/build, for example</clickurl>
  <tooltip>desired tooltip, Build 1245 passed, for example</tooltip>
</streamicon>"
```

setProperty

The preceding example is formatted to make it easy to read, but you should enter it as a single string when you specify the value for the **streamCustomIcon** property. For example:

```
"<streamicon><image>success</image><clickurl>http://example.com/build</clickurl><tooltip>Build 1245 passed</tooltip></streamicon>"
```

If you refer to an image that is unknown to AccuRev, AccuRev displays the "warning" icon and a tooltip like the following: *Unknown icon name "sucess" in reserved stream property streamCustomIcon.*

Considerations for Using XML

Note that the XML used to specify the `<streamicon>` argument must be enclosed in quotes, and it must be well-formed. Reserved characters like the ampersand (&) must be properly escaped (as `&` or `&`, for example). If there are errors in the XML, AccuRev displays an error like the following when you click the status icon to open the URL:

```
Invalid XML in reserved stream property streamCustomIcon.
```

You might see this error if you do not properly escape an ampersand (&) in a URL parameter. In this case, AccuRev displays the warning icon and the error message shown here instead of those specified in the `<image>` and `<tooltip>` tags, respectively.

```
Illegal character in query at index 29:http://example.com/build?foo= bar
```

You might see this error if you do not properly escape a space in the URL specified in the `<clickurl>` tag. Note that in this case, AccuRev displays the icon and tooltip you defined (assuming they have been correctly specified).

Setting Custom User Properties

AccuRev supports the use of an XML file to define custom user properties that display in the user dialogs. The file name for custom properties is `customUserProps.xml` or any name that you give it. Format the file as demonstrated in the following example in order to define custom fields.

```
<fields>
  <field name="Display Name"/>
  <field name="Phone Number"/>
  <field name="Email"/>
</fields>
```

Formatting the Tooltip

AccuRev supports the use of HTML tags to format tooltips displayed using the `<tooltip>` tag. For example, you can:

- Format tooltip text as bold or italic (using `` or `<i>` tags)
- Create new lines (using `
` or `<p>` tags)
- Create lists (using `` and `` tags)

If you want to use HTML to format **streamCustomIcon** tooltips, you need to:

- Specify all formatting between `<html>` and `</html>` tags
- Specify less-than and greater-than brackets using `<` and `>`, respectively, for both the `<html>` tags and any formatting tags they contain

For example, if you wanted to display this tooltip:

setproperty

```
Build 1798 complete
12 warnings
3 errors
```

you would specify the `<tooltip>` content as follows:

```
"<streamicon>
<tooltip>&lt;html&gt;Build 1798 complete&lt;br&gt;12 warnings&lt;br&gt;3 errors
&lt;/html&gt;</tooltip>
</streamicon>"
```

Setting Staging Stream Status

If you are using gated streams in your stream hierarchy, there might be occasions when you want to manually override the Running status on a staging stream. For example, if your software development process requires a code review before changes can be promoted, your reviewer needs a way to set the staging stream status when the review is complete.

You can manually set the staging stream status using the **stagingStreamResult** reserved stream property:

```
accurev setproperty -s <stream name> -r stagingStreamResult "status"
```

If the stream name contains spaces, you must enclose it in quotation marks. Valid values for *status* are "running", "success", "failed", and "warning".

Examples

On stream "prod_2010", set a property named "model" with a value of "obsolete":

```
> accurev setproperty -s prod_2010 model obsolete
```

On stream "nightly", set the "streamStyle" property to red:

```
> accurev setproperty -s nightly -r streamStyle "<style><color><background-color>#FF0000</background-color></color></style>"
```

On user "jsmith", set a property named "contractor" with a value of "AcmeTemps":

```
> accurev setproperty -u jsmith contractor AcmeTemps
```

Note that the order of the name/value pair changes to value/name when using the `-l` option to specify a file as a property value, as shown in the following two examples:

On user "jsmith", set a property named "profile" with a value of "jsmith.png":

```
> accurev setproperty -u jsmith -l jsmith.jpg profile
```

On user "jsmith", change the "profile" property to "jsmith_smiling.png":

```
> accurev setproperty -u jsmith -l jsmith_smiling.jpg profile
```

Set the status of staging stream **acme dev** to success:

```
> accurev setproperty -r -s "acme dev" stagingStreamResult "success"
```

See Also

[getproperty](#), [rmproperty](#), [show](#)

show list objects of a particular kind

Usage

```

show [ -fx ] accuwork
show [ -fx ] depots
show [ -f<format(s)> ] [ -u <principal> ] groups
show [ -fx ] locks
show [ -fx ] [ -g <group-name> ] members
show [ -f<format(s)> ] [ -ks [ -p <depot> ] | -ku | -s <stream> |
-u <principal> ] [ -n <pattern> ] [ -r ] properties
show [ -f<format(s)> ] refs
show [ -fx ] sessions
show [ -f<format(s)> ] slices
show [ -f<format(s)> ] [ -p <depot> ] [ -d ] [ -t <time-spec> ]
[ -s <stream> [ -m[<match-type>] ] [ -r | -R | -1 ] ] streams
show [ -f<format(s)> ] [ -p <depot> ] [ -d ] [ -t <time-spec> ]
[ -l <list-file> [ -m[<match-type>] ] [ -r | -R | -1 ] ] streams
show [ -fx ] [ -p <depot> ] -s <depot> stream_rules
show [ -f<format(s)> ] [ -p <depot> ] triggers
show [ -f<format(s)> ] users
show [ -f<format(s)> ] [ -a ] wspaces

```

Description

The **show** command displays information about all objects of a particular type:

- **accuwork**: Lists the depots in the repository that contain AccuWork issues. Displays the same information as **depots**.
- **depots**: For each AccuRev depot, displays (**Depot**) the depot name, (**Depot#**) the depot number, and (**Slice#**) the slice number.
- **groups**: For each group, displays (**Group#**) the group number and (**Group**) the group name. If a *<principal>* is specified, displays all the groups to which that principal belongs.
- **locks**: Displays a line for each stream lock, in this form:
 - {to|from|all} <stream-name> [{except|only} for <user-or-group-name>] [<comment>]
- **members** (of groups): For each user who belongs to any group — or to the particular group specified with the **-g** option — displays (**User**) the user's principal-name and (**Group**) the group name.
- **properties**: Displays properties created using the [setproperty](#) command for the specified stream or principal.
- **refs** (reference trees): Similar to the **wspaces** display (see below). The **Stream#** value identifies the stream or snapshot that the reference tree is based on. The workspace type value (next-to-last number) is always 3.
- **sessions**: Lists all the active login sessions for your AccuRev Server, indicating the IP address of the user's client machine and the duration of the session (in minutes).

show

- **slices:** For each AccuRev depot, displays (**Slice#**) the slice number, and (**Location**) the full pathname to the directory within the repository that stores the data for that depot.
- **streams:** For each stream in the repository — or a subset of the streams specified by the **-p**, **-s**, **-R**, **-r**, **-l**, **-d**, **-m**, **-l**, and/or **-t** options — displays:
 - (**Stream**) the stream name
 - (**Backing Stream**) the name of the stream’s backing/parent stream
 - (**Depot**) the name of the depot to which the stream belongs
 - (**Stream#**) the stream number
 - (**Dyn**) a “Y” if the stream is dynamic
 - (**Basis Time**) the basis time, if any, for a dynamic stream

The listing includes all kinds of streams, including workspace streams, pass-through streams, and snapshots.

startTime and time in -fx output -- If you use the **-fx** option for **show streams**, the XML output includes a *startTime* value for all streams, which specifies the time the stream was created. If the stream is a snapshot, the **show -fx streams** XML output also includes a *time* value, which specifies the time the snapshot was created. Both times are written in UNIX epoch format.

- **stream_rules:** For the stream specified by the **-p** and **-s** options, lists the workflows and workflow rules associated with the stream that dictate:
 - What workflow an issue must be in before it can enter or leave a stream
 - Optionally, what workflow stage an issue must be in before it can enter or leave a stream
 - Whether or not to allow incomplete issues or individual files to be promoted.

By default, output is in XML format, regardless of whether or not the **-fx** option is used.

- **triggers:** For each trigger created with the **mktrig** command, displays the type of trigger and the name or pathname of the trigger executable (script or program). Use the **-p** option to limit the listing to a particular depot’s triggers. Triggers that are enabled by creating a script at a well-known pathname (e.g. **server_admin_trig**) are not listed by this command.
- **users:** For each user, displays (**User#**) the unique user-ID number and (**User**) the principal-name.
- **wspaces** (workspaces): For each workspace that belongs to you — or for all workspaces in the repository — displays
 - (**Name**) the workspace name
 - (**Storage**) the full pathname to the workspace tree
 - (**Host**) the name of the machine where the workspace tree resides
 - (**Stream**) the stream number of the workspace stream, and four additional items:
 - (**Target_trans**) how up-to-date the workspace *should* be.
 - (**Trans**) how up-to-date the workspace actually is. Usually, Target_trans and Trans levels are the same. See [Update Level](#) in the [update](#) reference page.
 - (**depot**) the depot with which the workspace is associated.
 - (**fileModTime**) the timestamp of the last time the workspace was updated, converted to hex.

show

- **(Type)** one of the following: 1 (standard workspace), 9 (exclusive-file locking), 17 (anchor-required).
- **(EOL)** One of the following: 0 (platform-appropriate), 1 (UNIX/Linux style: NL), 2 (Windows style: CR-LF)
- **(user_id)** the unique user-ID number
- **(user_name)** the principal-name.

Options

- 1** (“dash-one”) Also lists the immediate children of the specified stream(s) when you use *show streams* with the **-s** or **-l** option.
- a** Display all workspaces. By default, *show* displays only workspaces that belong to you (i.e. to your principal-name).
- d** Restrict the display to streams with non-empty default groups.
- f <format(s)>**
Use one or more of the following format letters:
 - i**: Include deactivated (removed) items in the listing; for properties, include properties for hidden streams or principals.
 - I**: Include deactivated (removed) items, and include old definitions of all items. Cannot be used with the **-d** option, nor with the **-l**, **-m**, **-r**, or **-R** options in *show streams*.
 - v**: For *show users*, adds a Kind column to output, indicating whether the user is licensed for use of “full” (both configuration management and issue management) or “dispatch” (issue management only). For *show wspaces*, adds a Loc (location) column to output.
 - x**: Display the results in XML format.
 - xg**: (valid for *show streams*) Display the results in XML format. This variant includes the **hasDefaultGroup** attribute, which indicates whether each stream has a default group. Requires the **-p** option.

Note: To combine two or more formats, include them in the same expression. For example to display a list of deactivated items in XML format, specify **-fxi**. If you repeat the **-f** option (**-fx -fi**), the second instance is ignored.
- g <group-name>**
Restrict a member listing to the specified group.
- ks | ku**
Indicates the type of properties to display: stream (**-ks**) or user (**-ku**). By default, *show properties* displays stream properties if **-p**, **-s**, or no options are specified; user properties are displayed if the **-u** option is specified.
- l <list-file>**
Process the streams listed in the specified file. The file must be text only, containing one stream name or number per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing whitespace around the filenames. There is no provision for comment lines in a list-file. This option can also be used with one of **-l** (“dash-one”), **-r**, or **-R**. You cannot also specify streams using the **-s** option.

show

-m <match-type>

Indicates how the character string(s) specified with **-s** or **-I** are to be used in selecting streams to be listed. You can use any one of the following “match types”:

-mx: (default, if **-m** not specified) Selects a stream if its name exactly matches the string specified with **-s** or any of the strings in the list-file specified with **-I**.

-mc: Like **-mx**, but a specified string can occur anywhere within a stream name.

-me: Like **-mx**, but a specified string must match the end of a stream name.

-ms: Like **-mx**, but a specified string must match the start of a stream name.

With **-mx** or no **-m** option, matches are made with both active and removed streams; with all other **-m** options, matches are made only with active streams, unless you also specify the **-fi** option.

-n <pattern>

For **show ... properties**, specifies the glob pattern the property name must match; only properties whose name matches the glob pattern are displayed.

The glob pattern is a simple pattern that allows "*" to specify one or more unmatched characters or "?" to match exactly one unmatched character. Use the backslash (\) as an escape character to pass a literal "*" or "?". A name with no pattern characters will be exactly matched.

Note: You must use **-n** to specify a pattern for the property name if you use the **-I** option.

-p <depot>

Restrict a stream or trigger listing to the specified depot. The depot can be specified by name or number. Required if you are using the **-ks** option with **show ... properties**.

-r Also list the stream path and all children of the selected stream(s). Must be used in combination with the **-s** or **-I** option.

For **show ... properties**, **-r** shows the value of the reserved **streamStyle** property.

-R Also list all children of the selected stream(s). Must be used in combination with the **-s** or **-I** option.

-s <stream> or **-s** <stream-pattern>

(*show streams* only) Restrict the scope to the specified stream — or streams matching the stream pattern that you specify with **-m**. You can use **-I** (“dash-one”), **-r**, or **-R** along with this option. The stream can be specified by name or number. If you specify a stream by number, you must also specify a depot with the **-p** option.

(*show stream_rules* only) Restrict the scope to the specified stream. The stream can be specified by name or number. If you specify a stream by number, you must also specify a depot with the **-p** option.

In either case, if you use **-s**, you cannot also specify a list of streams (or stream-patterns) using the **-I** option.

-t <time-spec>

Show the streams that existed at the specified time. You must also use one of the **-I**, **-s**, or **-p** options. Note that a renamed stream may appear under its new or old name, depending on the <time-spec> you specify.

A time-spec can be any of the following:

show

- Time in <"YYYY/MM/DD HH:MM:SS"> format, "2007/08/07 20:27:15", for example. Note that you must use quotes when specifying time-spec as YYYY/MM/DD HH:MM:SS to ensure that the entire argument is interpreted by the command shell as a single token.
- Time keyword: **now**
- Transaction number as a positive integer: e.g. **146** or **23965**
- Transaction number keyword: **highest**

When using the **-t** option, take into account the possibility of a timewarp. The AC_SYNC environment variable (see [AccuRev User Preferences](#) on page 6) determines how a timewarp will be handled. See also section [System Clock Synchronization](#) on page 31 of the *AccuRev Administrator's Guide*.

-u <principal>

For **show ... properties**, the AccuRev user whose properties are to be displayed.

For **show ... groups**, displays all the groups to which the specified principal belongs.

Examples

Display the names of all depots in the repository:

```
> accurev show depots
```

Display the principal-names of all AccuRev users:

```
> accurev show users
```

Display the value of the **streamStyle** property for the stream **nightly**:

```
> accurev show -s nightly -n streamStyle -r properties
```

Display the names of all streams in depot **gizmo**:

```
> accurev show -p gizmo streams
```

Display the names of the streams that existed in depot **gizmo** at the time of transaction 248:

```
> accurev show -p gizmo -t 248 streams
```

Display the name of the stream **renamed_stream** (renamed in transaction 100) at the time of transaction number 50:

```
> accurev show -t 50 -s renamed_stream streams
```

Display the stream path and all children for stream **devt_brass** in depot **gizmo**:

```
> accurev show -p gizmo -s devt_brass -r streams
```

Display the workflow rules for stream **devt_brass** in depot **gizmo**:

```
> accurev show -fx -p gizmo -s devt_brass stream_rules
```

```
<?xml version="1.0" encoding="utf-8"?>
<AcResponse Command="show stream_rules">
  <Stream Name="devt_brass" allow_incomplete_issues="yes">
    <EntryRules>
      <Workflows>
        <Workflow Name="defects" restrict_stages="no">
          <Transitions>
            <Transition from_stage="WIP" to_stage="New"/>
          </Transitions>
        </Workflow>
      </Workflows>
    </EntryRules>
  </Stream>
</AcResponse>
```

show

```
        <Transition from_stage="Dev Complete" to_stage="Deferred"/>
    </Transitions>
</Workflow>
<Workflow Name="workflow_check_conditions" restrict_stages="no">
    <Transitions>
        <Transition from_stage="New" to_stage="Scheduled"/>
    </Transitions>
    <Transitions>
        <Transition from_stage="Scheduled" to_stage="WIP"/>
        <Transition from_stage="WIP" to_stage="Dev Complete"/>
    </Transitions>
</Workflow>
</Workflows>
</EntryRules>
<ExitRules>
    <Workflows>
        <Workflow Name="enhancements" restrict_stages="no">
            <Transitions>
                <Transition from_stage="Deferred" to_stage="Closed"/>
                <Transition from_stage="Dev Complete" to_stage="Deferred"/>
            </Transitions>
        </Workflow>
    </Workflows>
</ExitRules>
</Stream>
</AcResponse>
```

See Also

getproperty

sitehist show transaction history of non-depot specific site information

Usage

```
sitehist [ -t <transaction-range> ] [ -c <comment> ] [ -a ] [ -u <principal-  
name> ] [ -k <transaction-kind> ] [ -f <format(s)> ]
```

Description

The **sitehist** command displays the transaction history of non-depot specific site information such as stream acls, users and group membership.

Options

-a Return information about all site transactions.

-c <comment-string>

Display only transaction(s) whose comments contain the specified string. (The string matching is case-insensitive.) Enclose the string in quotes to ensure that **sitehist** interprets it as a single command-line argument.

-f <format(s)>

Use one or more of the following format letters:

t: ("transaction") Display just the header line, including the transaction number.

x: ("XML") Display results in XML format.

-k <transaction-kind>

Display only the transactions of the specified kind. Valid values are:

dbupgrade, set_acl, clear_acl, mkuser, chuser, chpasswd, remove_user, reactivate_user, mkgroup, chgroup, addmember, rmmember, remove_group, reactivate_group

-u <principal-name>

Display only the transactions for the specified AccuRev user.

-t <time-spec>[.<count>]

-t <time-spec> - <time-spec>[.<count>]

The first form specifies one transaction, or a specified number of transactions up to (i.e. preceding) and including a particular transaction.

The second form specifies transactions that took place in the specified interval; the optional suffix truncates the listing after the most recent <count> transactions. You may need to use quotes in composing the argument following -t; the entire argument must be interpreted by the command shell as a single token. The command output depends on the order of the time-specs; list the most recent time-spec first to show the results in order from most recent to least recent. Reverse the order of the time-specs to reverse the order of the output.

A time-spec can be any of the following:

sitehist

- Time in <"YYYY/MM/DD HH:MM:SS"> format, "2007/08/07 20:27:15", for example. Note that you must use quotes when specifying time-spec as YYYY/MM/DD HH:MM:SS to ensure that the entire argument is interpreted by the command shell as a single token.
- Time keyword: **now**
- Transaction number as a positive integer: e.g. **146** or **23965**
- Transaction number keyword: **highest**

When using the **-t** option, take into account the possibility of a timewarp. The AC_SYNC environment variable (see [AccuRev User Preferences](#) on page 6) determines how a timewarp will be handled. See also section [System Clock Synchronization](#) on page 31 of the *AccuRev Administrator's Guide*.

Specifying the Transactions

By default, **sitehist** displays the entire transaction history of the site. The various forms of the **-t** option restrict the display to a single transaction, or to a range of transactions. Each variant involves one or two time-specs. A single time-spec means "the transaction that took place at this time" or "the most recent transaction that took place before this time". Two time-specs define an interval; **sitehist** reports all the transactions in the interval.

Examples

Display the transaction history for the site:

```
> accurev sitehist
```

Display site transactions 145 through 176, inclusive:

```
> accurev sitehist -t "176-145"
```

Display the five most recent site transactions:

```
> accurev sitehist -t now.5
```

Display information on the most recent **mkuser** site transaction:

```
> accurev sitehist -t now -k mkuser
transaction 10; mkuser; 2016/03/15 16:14:09; user: u1
new; id: 10; name: u5_nopass; kind: full
```

Note that the output of this command can be in XML format if you specify **-fx** at the command line:

```
> accurev sitehist -t now -k mkuser -fx
```

```
<?xml version="1.0" encoding="utf-8"?>
<AcResponse
  Command="sitehist"
  TaskId="1301"
  <transaction
    id="10"
    type="mkuser"
    time="1458072849"
    user="u1">
  <User
    Info="new"
```


sitehist

```
        Number="10"  
        Name="u5_nopass"  
        Kind="full"/>  
    </transaction>  
</AcResponse>
```

The <User> element details the number, name and kind of user that was created by the **mkuser** transaction. A transaction's timestamp (time attribute of the XML element <transaction>) is reported as the number of seconds since Jan. 1, 1970 UTC. Special Field Types on page 264 describes a technique for converting this timestamp into a human-readable string.

The user attribute of the <transaction> element displays the name of the user who issued the **mkuser** transaction.

Display information on the most recent **mkgroup** site transaction:

```
> accurev sitehist -t now -k mkgroup -fx  
<?xml version="1.0" encoding="utf-8"?>  
<AcResponse  
    Command="sitehist"  
    TaskId="1302">  
    <transaction  
        id="29"  
        type="mkgroup"  
        time="1458072851"  
        user="u1">  
        <Group  
            Info="new"  
            Number="17"  
            Name="gmeta2"/>  
        </transaction>  
    </AcResponse>
```

Display information on the most recent **chuser** site transaction:

```
> accurev sitehist -t now -k chuser -fx  
<?xml version="1.0" encoding="utf-8"?>  
<AcResponse  
    Command="sitehist"  
    TaskId="1305">  
    <transaction  
        id="51"  
        type="chuser"  
        time="1458073304"  
        user="u1">  
        <User
```

sitehist

```
        Info="old"
        Number="2"
        Name="u2"
        Kind="full"/>
    <User
        Info="new"
        Number="2"
        Name="testuser2"
        Kind="full"/>
</transaction>
</AcResponse>
```

Notice that there are two user elements in the transaction. The record with the Info="old" attribute contains the number, name and kind of user before the changes made by the **chuser** command. The element with the Info="new" attribute contains the new number, name and kind. Notice that the name changed from "u2" to "testuser2".

Display information on the most recent **set_acl** transaction:

```
> accurev sitehist -t now -k set_acl -fx
<?xml version="1.0" encoding="utf-8"?>
<AcResponse
  Command="sitehist"
  TaskId="1306">
  <transaction
    id="50"
    type="set_acl"
    time="1458072889"
    user="u1">
    <Acl
      action="set"
      depot="d1"
      stream=""
      user="gmeta2"
      allow_access="yes"
      inheritable="yes"/>
    </transaction>
  </AcResponse>
```

start

start create a command shell in a workspace or reference tree

Usage

```
accurev start { -w <workspace> | -r <reftree> }
```

Description

The **start** command starts a new command shell:

- UNIX/Linux: a subshell of the current shell.
- Windows: a new Command Prompt window.

In the new command shell, the current working directory is set to the top-level directory of the specified workspace or reference tree. In addition, these environment variables are set in the new command shell:

ACCUREV_WSPACE

The name of the workspace.

ACCUREV_TOPDIR

The full pathname of the workspace's top-level directory.

Using the **start** command is a convenience, not a requirement. An alternative, you can simply **cd** to any directory within the workspace. (This won't set the environment variables, though!) The **accurev** program automatically detects the fact that your current working directory is within an AccuRev workspace.

Options

-w <workspace>

Specify the workspace in which you wish to work. You don't need to include the “_<principal-name>” suffix in the workspace name.

-r <reftree>

Specify the reference tree in which you wish to work.

Examples

Start working in workspace **tulip_dvt**:

```
> accurev start -w tulip_dvt
```

See Also

mkws, **show** (wspaces)

stat

show the AccuRev status of elements or file system objects

Usage

Status of elements in your workspace:

```
accurev stat [ -f<format(s)> ] [ <element-selection-option> ] [ -B ] [ -L ] [
-M ] [ -O ] [ -R ] [ -t <time-spec> ] [--use_time_basis_for_overlap]
[ [ -Fx ] -l <list-file> | <element-list> ]
```

Status of external files in your workspace:

```
accurev stat -x [ -f<format(s)> ] [ -R ] [ --ignore ]
[ [ -Fx ] -l <list-file> | <element-list> ]
```

Status of versions in a specified stream:

```
accurev stat { -b | -s <stream> } [ -f<format(s)> ] [ -adDio ] [--
use_time_basis_for_overlap] [ -t <time-spec> ] [ [ -Fx ] -l <list-file> |
<element-list> ]
```

element-selection options:

```
-a -d -D -e -i -k -L -m -n -o -p -U -x -X --ignore --outgoing
```

-f option format letters:

```
[ i | I ] [ a | r ] [ f | d ] [ l ] [ e ] [ x ] [ k ] [ n ] [ o ]
```

Description

The **stat** command displays the status of files in your workspace, or of elements in a specified stream. The most basic use of **stat** is to show the current version of an element in your workspace stream. It can also be used to find out which elements in your workspace you have modified or which elements are ready to be promoted.

The **stat** listing includes:

name of element

The pathname of the element, relative to the depot's top-level directory.

virtual version

The virtual version-ID of the version currently in the stream.

(real version)

The real version-ID (enclosed in parentheses) of the version currently in the stream. See [Promotion: Real Versions and Virtual Versions](#) on page 4 of the *AccuRev Concepts Manual* for a detailed description of virtual versions and real versions.

status indicators

One or more of the keywords listed in the following section.

Status Indicators

An element's status consists of one or more of the following status indicators.

stat

Link-related indicators:

- **(elink)** — the element is an element link.
- **(slink)** — the element is a symbolic link.

(There is no status indicator for a file element or directory element.)

- **(missing-target)** — For an element link, the target element is not present in the workspace or stream. This can occur if the target element is removed from the workspace tree by an operating system command. It can also result from an *incl -b* or *excl* command. For a symbolic link, there is no object at the target pathname.
- **(modified-target)** — For an element link, the target element has been modified (either a content change or a namespace change) in the workspace or stream.
- **(defunct-target)** — For an element link, the target element has **(defunct)** status in this workspace or stream.
- **(nonexistent-target)** — For an element link, the target element does not appear at all in this stream or workspace. This occurs when the target element is *defunct*'ed then *promote*'d to the parent stream.
- **(corrupted)** — For an element link in a workspace, AccuRev and the operating system disagree on the link target. That is, the target element recorded in the AccuRev repository differs from the target in the operating system's instantiation of the link in the workspace tree. This can occur if you modify or replace a link using operating system commands instead of AccuRev commands.

A cross-linked element (see **(xlinked)** below) gets corrupted status if AccuRev does not overwrite the element during execution of the *Include from Stream* command, because the element has **(modified)** or **(kept)** status in the workspace. This should not occur during normal operation.

Presence of the element in the workspace:

- **(defunct)** — the element has been marked for removal from the workspace stream with the *defunct* command. The element has already been removed from the workspace tree (local disk storage); it gets removed from the workspace stream (in the depot) when you *promote* the element to the backing stream.
- **(external)** — the file or directory has not been placed under version control. (It's in the workspace tree, but not in the workspace stream.) See [Specifying Ignore Patterns for External Objects](#) on page 7 in *AccuRev Technical Notes*.
- **(excluded)** — the element does not appear in the workspace because it has been excluded, using the Include/Exclude facility. For file elements, it's more likely that the exclusion was explicitly set on the directory in which the file resides, or in a higher-level directory that includes the file. See the [incl](#), [excl](#), and [incldo](#) reference pages.
- **(xlinked)** — this version of the element appears in the workspace or stream by virtue of a cross-link (*incl -b* command) — either on the element itself or on its parent directory or a higher-level directory.
- **(missing)** — the workspace “should” include a version of this element, but doesn't. This occurs when you delete version-controlled files from the workspace tree using operating system commands. If an operation causes the target of an element link to be removed from a workspace, AccuRev removes the element link, also, causing it to have **(missing)** status.
- **(twin)** — the element is one of multiple elements in the workspace that exist at the same pathname. At most one of these elements can be accessed through the pathname; the other(s) can be accessed

stat

through their unique element-IDs. See [Version Control of Namespace-Related Changes](#) on page 37 in *AccuRev Technical Notes*.

- **(stranded)** — the element is active in the workspace, but cannot be accessed through the file system. This can occur in several situations:
 - There is no pathname to the element, because the element’s directory (or a higher-level directory) was removed from the workspace or stream.
 - (dynamic stream only) There are one or more defunct elements at a given pathname, along with one non-defunct element. The defunct element(s) have **(stranded)** status.
 - The element’s directory (or a higher-level directory) is cross-linked, making another version appear at the pathname of the active version.

Changes to the element in the workspace:

- **(modified)** — the file has been modified in the workspace since the most recent *update* or *keep*. See [Optimized Search for Modified Files](#) below.
- **(kept)** — a new version of the element has been created with *keep*, *move*, *defunct*, or *undefunct*, and the file has not subsequently been modified, *promote*’d to the backing stream, or *purge*’d.
- **(member)** — the element is “active” in the workspace (is in the workspace stream’s default group). The commands that create a new version, listed above, also make the element active. So do the commands *anchor*, *co*, and *revert*.

Relationship to the version in the backing stream:

- **(backed)** — the version in the workspace stream is the same as the version in the backing stream. And you have not changed the element since the last time you *promote*’d or *purge*’d it, or since the most recent *update* of your workspace.
- **(stale)** — the element needs to be updated, because the version in the backing stream has changed since the workspace’s latest *update*. And since you have not changed the element in your workspace, it can be safely updated.
- **(overlap)** — the element has changed both in the backing stream and in your workspace. This indicates that a merge is required before you can promote your changes to the backing stream. Prior to AccuRev 4.6, “underlap” files were considered to have “overlap” status.
- **(underlap)** — similar to *overlap*: the element has changed both in the backing stream and in your workspace, but the changes in your workspace have already been promoted to the backing stream. (More precisely, your version is an ancestor of the backing stream’s version.) In many cases, the most appropriate course of action is to *purge* the changes from your workspace, restoring the version that was in the backing stream at the time of the workspace’s most recent *update*. In other cases, a merge-promote sequence is most appropriate. Prior to AccuRev 4.6, “underlap” files were considered to have “overlap” status.

Depot-Relative Pathnames

A depot implements a (version-controlled) directory tree; thus, every element in the depot has a pathname within that directory tree. For example, depot **gizmo** might contain a top-level directory named **src**, which contains a subdirectory named **commands**, which contains a file name **base.h**. The element’s pathname within the depot is:

```
src/commands/base.h
```

stat

(Adjust the slashes to suit your operating system.) By default, **stat** lists files and directories using such depot-relative pathnames. It uses the distinctive prefix `./.` (UNIX/Linux) or `\.\` (Windows) to indicate a depot-relative pathname:

```
./src/commands/base.h      (UNIX/Linux depot-relative pathname)
\.\src\commands\base.h    (Windows depot-relative pathname)
```

stat can also list files using absolute pathnames (**-fa** option):

```
/home/jsmith/gizmo_dvt_jsmith/src/commands/base.h  (UNIX/Linux absolute pathname)
c:\gizmo_dvt_jsmith\src\commands\base.h           (Windows absolute pathname)
```

Likewise, **stat** can list files using relative pathnames (**-fr** option). If the current working directory is **src**, then the relative pathname of file **base.h** is:

```
./commands/base.h        (UNIX/Linux relative pathname)
.\commands\base.h       (Windows relative pathname)
```

Optimized Search for Modified Files

When **stat** searches your workspace for files that have been modified (with any of the options **-n**, **-m**, **-p**, **-o**) or that have not yet been added to AccuRev (**-x**), it must scan the entire workspace on your hard drive. This might involve many thousands of files, and is potentially quite time-consuming. In these searches, **stat** defaults to using a timestamp optimization to speed its search for modified files: it does not consider files whose timestamps have not changed since the last time the workspace was updated or searched for modified files.

Use the **-O** option to have **stat** dispense with the optimization and consider all files, regardless of timestamp, in its search for modified files.

Options

- a** Select all elements in the stream. You cannot also specify a list of elements, either on the command line or with a list-file (**-l**).
- b** Display the status of the version in the backing stream, not the file in the workspace. See also **-s**. **Note:** To handle files that have been renamed in the backing stream, **stat** first looks up the specified filename in the workspace, gets the EID, and uses the EID to find the correct file in the backing stream. If the specified name is not found in the workspace, then the backing stream is searched. In any case, the name of the file as it currently exists in the backing stream is what gets displayed.
- B** Also display the status of all versions in the backing chain of streams above the workspace stream. This reveals any deep overlaps or underlaps for the element.
- d** Select only active elements (those in the default group) of the workspace or stream.
- D** Select only defunct elements in the workspace or stream.
- e <eid>**
Select the file element with the specified element-ID. You can use this option instead of specifying the name of an element. If specified, this option takes precedence over other options, such as **-a**. If you also specify a `<list-file>` or `<element-list>`, it is ignored.

-f <format(s)>

By default, *stat* displays both files and directories; for each one, it displays the location as a depot-relative pathname (see [Depot-Relative Pathnames](#) above), status indicators, the virtual version-ID, and the real version-ID in parentheses. Use one or more of the following format letters to restrict displays to either files or directories and to control other formatting:

i: Select all files and directories, even if they are excluded by the value of the **.acignore** file. (See the **-x** description, and [Specifying Ignore Patterns for External Objects](#) on page 7 of *AccuRev Technical Notes* .)

I: Select only those files and directories specified by the value of the **.acignore** file.

a: Display locations as absolute pathnames.

r: Display locations as relative pathnames (relative to the current working directory).

f: Display files only.

d: Display directories only.

l: Display locations only (no status indicators or version-IDs).

n: Split each element's listing onto two lines: the first contains the depot-relative pathname; the second contains the status information. (This option overrides all other **-f..** options.)

o: Display deep overlap elements that exist above a time basis stream. **Note:** The default behavior without **-fo** is to stop at a time basis (same as *stat -o -B*), to improve performance and return fewer irrelevant results.

When combined with the XML option (**-fox**), this option causes a "timeBasisStream=" attribute to be appended to the XML output for overlap elements that exist above the time basis stream.

v: Display the target of an element link or symbolic link. If an element link is part of a multiple-link chain, only the element at the end of the chain is displayed.

x: Display the results in XML format.

You can use appropriate combinations of the above options — for example, **-fda** or **-frl**. The following additional keyletters can be used in combination with each other, and with the ones above:

e: Display the element-ID.

k: Display the element type (that is, data type) of this version; different versions of an element can have different element types; see [Controlling the Element Type and Exclusive File Locking State](#) in the [add](#) reference page.

-i Select only stranded elements: members of the default group that no longer have a pathname in the workspace or stream.

-k Select only kept elements.

-L Allows you to specify the full pathname of a directory in an AccuRev workspace. If this option is omitted, AccuRev defaults to the current workspace.

-m Select only modified elements. Can be used with **-d**.

-M Restrict the listing to elements that are missing from the workspace tree.

-n Select only modified elements that are not in the workspace's default group.

-o Select only elements with overlaps in the workspace or stream, which require a merge. See also **-B**.

-O Override the optimized search for modified files. When **-O** is used, *all* files are considered, even those whose modification times have *not* changed since the last time the workspace was updated or

stat

searched for modified files. Having to check all files, regardless of modification time, slows *stat* performance. See [Optimized Search for Modified Files](#) above.

-p Select only elements that are pending promotion from the workspace or stream.

-R Process all the elements in the specified directory tree(s). Use “.” to specify the current working directory.

-s <stream>

Display the status of the version in the specified stream (or snapshot), not the file in the workspace. See the description of *<element-list>* below for notes on how to specify the element(s). See also *b*.

--use_time_basis_for_overlap

Use the workspace's update level or the stream's time basis to identify overlap instead of using the current time. When this option is used, AccuRev shows (**overlap**) status for a file only if the file was overlapped at the time indicated by the workspace's update level or the stream's time basis.

-U Select only underlap files and directories.

-x Select only external files and directories.

-X Display files and directories that have been excluded from the workspace or stream (or from a higher-level stream) with the *excl* command. These objects are listed with the status indicator (**excluded**).

--ignore

Specify a pattern of elements; external elements matching this pattern are ignored by the *stat* command. The **--ignore** option can be used multiple times for the same command, but each instance can specify only one pattern. For example:

```
accurev stat -x --ignore="*.obj" --ignore="*.txt" --ignore="!file1.txt"
```

Can also be used to specify the pathname of a text file that contains one or more patterns. For example:

```
accurev stat -x --ignore="@c:\work\my_ignore.txt"
```

See [Specifying Ignore Patterns for External Objects](#) on page 7 of *AccuRev Technical Notes* for more information on specifying ignore patterns for external files.

Note that the patterns expressed using the **--ignore** option take precedence over patterns expressed using the *.acignore* file.

--outgoing

Display all files in the workspace or stream with one or more of the following statuses: (**member**), (**modified**), (**missing**), or (**external**). Can be used with **-i** to return any (**stranded**) elements.

-t <time-spec>

Stat elements based on a point in time or a transaction number. A time-spec can be any of the following:

- Time in *<"YYYY/MM/DD HH:MM:SS">* format, "2007/08/07 20:27:15", for example. Note that you must use quotes when specifying time-spec as *YYYY/MM/DD HH:MM:SS* to ensure that the entire argument is interpreted by the command shell as a single token.
- Time keyword: **now**

stat

- Transaction number as a positive integer: e.g. **146** or **23965**
- Transaction number keyword: **highest**

When using the **-t** option, take into account the possibility of a timewarp. The `AC_SYNC` environment variable (see [AccuRev User Preferences](#) on page 6) determines how a timewarp will be handled. See also section [System Clock Synchronization](#) on page 31 of the *AccuRev Administrator's Guide*.

-Fx Signals that `<list-file>` (see the **-l** option) is an XML-format file, not a flat text file. Example:

```
<elements>
  <e eid="2"></e>
  <e eid="200"></e>
</elements>
```

-l <list-file>

Process the elements listed in `<list-file>`. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any `<element-list>` specified is silently ignored.

<element-list>

One or more element names, separated by whitespace. If you also specify a `<list-file>` using the **-l** option, this `<element-list>` is silently ignored.

You can specify patterns, using the wildcard characters `?` (match any character) and `*` (match any 0 or more characters).

If you use **-s** to display the status of a non-workspace stream, then each name in `<element-list>` is interpreted relative to the top-level directory of the depot. Typically, a simple filename like **base.h** won't be valid; instead, use a pathname like **src/include/base.h**. Alternatively, use a full depot-relative pathname, like **./src/include/base.h**.

Examples

List all modified elements in the workspace:

```
> accurev stat -m
```

List all modified elements in the workspace's default group:

```
> accurev stat -md
```

List all modified elements in the current directory:

```
> accurev stat -m *
```

List all elements in the default group of stream **tulip_test**:

```
> accurev stat -s tulip_test -d
```

List all **.doc** files in the current working directory, displaying the data-type of each one:

```
> accurev stat -ffk *.doc
```

List all elements in the stream as of transaction id **23564**:

```
> accurev stat -t 23564 -a
```

stat

List all elements in the stream as of January 5, 2010, at 1:00.:00:

```
> accurev stat -t "2010/01/05 01:00:00" -a
```

List all elements in **Stream1**'s default group as of transaction id **18543**:

```
> accurev stat -t 18543 -s Stream1 -d
```

List all elements in **Stream1**'s default group as of January 5, 2010:

```
> accurev stat -t "2010/01/05 01:00:00" -s Stream1 -d
```

List all deep overlap elements, even if they occur above a time basis stream, and return the results in XML:

```
accurev stat -s Stream_1 -o -B -fox
```

List all elements in stream **tbase1**, but do not show (overlap) status if the overlap was created at a higher transaction level than **tbase1**'s time basis.

```
accurev stat -s tbase1 --use_time_basis_for_overlap
```

See Also

add, files

Techniques for Selecting Elements on page 10

[The Update Algorithm](#) on page 26 of *AccuRev Technical Notes*

synctime

synctime

synchronize system clock on client computer to server computer

Usage

```
accurev synctime
```

Description

The *synctime* command changes the system clock on your host (the client computer) to match the clock on the host where the AccuRev Server is running (the server computer).

Note: if the client computer is running UNIX/Linux, you must be the **root** user.

Examples

Change your machine's system clock to match the clock on the machine running the AccuRev Server process:

```
> accurev synctime
```

See Also

[System Clock Synchronization](#) on page 31 of the *AccuRev Administrator's Guide*.

touch update the timestamp of a file

Usage

```
accurev touch [ -R ] [ -t <time-spec> ] <file-list>
```

Description

The **touch** command updates the time-last-modified timestamp on one or more files in your workspace. To set the timestamps, it uses the system clock on your host (the client computer) or the time you specify with **-t**.

Note: **touch** does not affect the depot at all, just the specified file(s) in your workspace.

Typically, you should use **touch** on files that you copy into the workspace from another location — your home directory, a remote FTP site, etc. Some applications update the timestamps on the files they copy; others don't.

Options

-R For each directory in *<file-list>*, process all the files in its subdirectory tree.

-t <time-spec>

A time-spec can be:

- Time in *<"YYYY/MM/DD HH:MM:SS">* format, **"2007/08/07 20:27:15"**, for example. Note that you must use quotes when specifying time-spec as *YYYY/MM/DD HH:MM:SS* to ensure that the entire argument is interpreted by the command shell as a single token.
- Time keyword: **now**

When using the **-t** option, take into account the possibility of a timewarp. The `AC_SYNC` environment variable (see [AccuRev User Preferences](#) on page 6) determines how a timewarp will be handled. See also section [System Clock Synchronization](#) on page 31 of the *AccuRev Administrator's Guide*.

Examples

Update the timestamps of two files:

```
> accurev touch test/rebase.sh doc/rebase.doc
```

Update the timestamps of all files in two subdirectories:

```
> accurev touch -R src lib
```

See Also

[add](#), [keep](#), [stat](#)

[Techniques for Selecting Elements](#) on page 10

trace-event monitor notification events from the server

Usage

```
accurev trace-event [ --category <categories> ]
[ -p <depots> ] [ -s <streams> ] [ -u <users> [ --exclude_users ] ]
[ -k <transaction-kinds> [ --exclude_kinds ] ] [ -f <formats> ]
[ -l <response> ]
```

Description

The **trace-event** command monitors server events by listening for messages published by the server. The server uses the Mosquitto message broker and MQTT (MQ Telemetry Transport) to publish a message for each write operation that it executes. By listening for these messages instead of polling, **trace-event** avoids creating any load on the server.

The **trace-event** options apply filters to constrain which events are displayed in the output. They also enable running an external application or script in response to the events. Multiple instances of **trace-event** can be run on a client machine (or VM) for different purposes.

The **trace-event** command continues to listen for events (and respond if applicable) until the application is terminated via Ctrl-C.

Options

Note: *If multiple values are specified for an option, separate the values with commas. If any of the values contains a space in it, place double quotes around the entire list of values.*

--category <categories>

Specify the categories of message events to display. If this option is not specified, **trace-event** displays all categories of events. Valid values are:
stream, accuwork, depot, site

Note: **--category <categories>** cannot be used together with **-k <kinds>**.

-p <depots>

Specify the depots to watch for in **depot**-category message events.

-s <streams>

Specify the streams to watch for in **stream**-category message events.

-u <users>

Specify the users to watch for in all categories of message events.

--exclude_users

Use in conjunction with the **-u** option to indicate that the named users should be filtered *out* and that messages for all other users should be displayed in the output.

-k <transaction-kinds>

Specify the kinds of transactions to display. Valid values are:

add, addmember, archive, backup, chstream, chws, clear, chdepot, chgroup, chpasswd, chuser, chref, chslice, co, cpkadd, cpkremove, defunct, demote, eacl, excl, incl, incldo, keep, ln, lock,

trace-event

mkdepot, mkgroup, mkrules, mksnap, mkstream, mkuser, mkref, mkreplica, mktrig, mkws, modify_issue, move, new_issue, promote, purge, putconfig, reactivate_group, reactivate_ref, reactivate_stream, reactivate_user, reactivate_wspace, reclaim, remove_group, remove_ref, remove_sessions, remove_stream, remove_user, replica_sync, rmmember, rmproperty, rmreplica, rmtrig, rmws, schema, setacl, setproperty, unarchive, undefunct, unlock

Note: **-k** <kinds> cannot be used with **--category** <categories>.

--exclude_kinds

Use in conjunction with the **-k** option to indicate that the named kinds of transactions should be filtered *out* and that messages for all other kinds of transactions should be displayed in the output.

-f <formats>

Use one or more of the following format letters:

-v: (“verbose”) Show the comment and timestamp for the transaction.

-x: (“XML”) Show the results in XML format.

-l <response>

Specify the path of an external application or script to be executed in response to each *trace-event* message.

The *trace-event* command does simple argument substitution to enable arguments to be passed to the response application:

\$depot – substitute the depot name

\$kind – substitute the transaction kind

\$principal – substitute the name of the user who executed the operation

\$members - substitute the user(s) or group(s) that are affected by the operation (if applicable)

\$stream – substitute the stream name (if applicable)

\$transnum – substitute the transaction number (if applicable)

\$issues - substitute the issue number(s) (if applicable)

\$comment – substitute the comments (if applicable)

For example, if the response option is “**-r /home/\$principal/logger.py \$comment**”, then **\$principal** is replaced with the message’s user name and **\$comment** is replaced with the message’s comment.

Examples

Display *all* published messages:

```
> accurev trace-event
```

Display all **site**-related messages and ignore messages related to streams, depots, and AccuWork:

```
> accurev trace-event --category site
```

Display verbose messages published for depots **d1** and **d2**:

```
> accurev trace-event -p d1,d2 -fv
```

Display **promote** messages for streams **release** and **production**:

```
> accurev trace-event -s release,production -k promote
```

Display **promote** messages for all users *except* john smith and jones:

trace-event

```
> accurev trace-event -u "john smith,jones" --exclude_users -k promote
```

Display **promote** and **chstream** messages for the stream named **backing**. When such events occur, execute **"accurev update -m"** to merge the updates:

```
> accurev trace-event -s backing -k promote,chstream -l "accurev update -m"
```

Display **promote** messages for the stream named **backing**. When such events occur, run the python script **C:\tools\promote_response.py** and pass the depot name to the script:

```
> accurev trace-event -s backing -k promote -l "python C:\tools\promote_response.py $depot"
```


translist list transactions containing versions that need to be promoted

Usage

```
accurev translist [ -s <stream> ] [ -fx ]
```

Description

The *translist* command lists the numbers of the transactions — *keep*, *move*, *defunct*, etc. — that created the versions that are currently active in the specified stream (default: your workspace stream). Exception: *anchor/co* transactions are not listed.

For the purposes of this command, a version is active in a stream if it has been promoted to (or created in) the stream, but has not yet been promoted out of the stream to its parent (or *purge*'d from the stream). For example, if several *promote* transactions created successive versions of the same element in a stream and that element has not been promoted to the stream's parent, then *translist* will list all the successive *promote* transaction numbers.

Note: This is a more expansive definition of “active” than the standard “in the stream's default group” definition.

Some or all of the listed transactions might also include versions that have already been promoted; if a transaction contains even a single unpromoted version, *translist* includes it.

Options

-s <stream>

Specify the stream whose versions need to be promoted. If you omit this option, *translist* uses your workspace stream.

-fx Display the results in XML format.

Examples

List all the transactions that created versions in your workspace stream, but which you haven't yet promoted to the backing stream:

```
> accurev translist
```

See Also

hist, *mergelist*, *patchlist*

unarchive restore version container files that were previously archived

Usage

```
accurev unarchive [ -E <element-type(s)> ] [ -i ]
  [ -s <stream> ] [ -t <transaction-range> ]
  [ -R ] { [ -Fx ] -l <list-file> | <element-list> }

accurev unarchive -a { [ -Fx ] -l <list-file> | <element-list> }
```

Description

The *unarchive* command restores to a depot's file storage area one or more version container files that were previously archived. The container files shift from *archived* status to *normal* status. The container files are moved from the depot's gateway area back to their original locations in the depot's file storage area.

You can use this command in either of these ways:

- To restore the versions that were archived by one particular invocation of *archive*, use command-line options that match the options specified in the original *archive* command. (But don't include any comment specified in the original command with *-c*.) For example:

- command to archive versions:

```
accurev archive -R -t 34591 -c "all deliverable files" binaries
```

- command to restore archived versions:

```
accurev unarchive -R -t 34591 binaries
```

- To restore all archived versions of a particular set of elements, use the *-a* option. For example:

```
accurev unarchive -a -l archived_elements.2007_03_14
```

Options

-i List the versions that would be unarchived (in XML format), but do not actually unarchive them.

-E <element-type(s)>

Restrict the unarchive operation to elements of one or more element-types: **text**, **binary**, and/or **ptext**. To specify multiple types, use a comma-separated list (default: **all** specifies all element types). Examples:

```
-E ptext,binary
-E "text, ptext"
```

-s <stream>

Restrict the unarchive operation to versions in the specified stream (or snapshot, or workspace). Default: unarchive versions in the current workspace stream.

If you use this option, you must identify the elements to be processed using depot-relative pathnames.

-t <transaction>[.<count>]

unarchive

-t *<transaction>* - *<transaction>*[*.<count>*]

The first form specifies one transaction, or a specified number of transactions up to (i.e. preceding) and including a particular transaction, for example: **23695** or **now.5**.

The second form specifies a range of transactions; the optional suffix truncates the listing after the most recent *<count>* transactions. You may need to use quotes in composing the argument following **-t**; the entire argument must be interpreted by the command shell as a single token. You cannot use the **now** or **highest** keyword in the interval specification.

For the **unarchive** command, a transaction can be represented in the following ways:

- Time keyword: **now**
- Transaction number as a positive integer: e.g. **146** or **23965**
- Transaction number keyword: **highest**

-R Recurse into each directory specified in *<element-list>* or *<list-file>*.

-Fx Signals that *<list-file>* (see the **-l** option) is an XML-format file, not a flat text file. Example:

```
<elements>
  <element location="..\dir07\sub01\file01.txt"/>
  <element location="..\dir07\sub01\file02.txt"/>
  <element location="..\dir07\sub01\file14.txt"/>
</elements>
```

The **-Fx** option must precede the **-l** option on the command line.

-l *<list-file>*

Process the elements listed in *<list-file>*. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any *<element-list>* specified is silently ignored.

<element-list>

One or more element names, separated by whitespace. If you also specify a *<list-file>* using the **-l** option, this *<element-list>* is silently ignored.

See Also

[archive, reclaim](#)

[Techniques for Selecting Elements](#) on page 10

[Archiving of Version Container Files](#) on page 37 of the *AccuRev Administrator's Guide*.

undefunct restore a previously removed element to a stream

Usage

```
accurev undefunct [ -c <comment> ] { <element-list> | -e <eid> |
    -l <list-file> }
```

Description

The *undefunct* command undoes the effect of the *defunct* command: it restores one or more elements that had previously been removed from your workspace. The backing stream’s version of each element is copied to the workspace.

As with all changes to the workspace, you must use *promote* to send the change made by *undefunct* (restoration of a removed element) to the backing stream.

Defuncting Directories

If any of the elements you specify is a directory, *undefunct* works recursively: it restores the directory itself and all elements under that directory.

Exclusive File Locking and Anchor-Required Workspaces

See [File Locking in Workspaces](#) on page 5.

Options

-c <comment>

Specify a comment for the transaction. The next command-line argument should be a quoted string. Alternatively, the next argument can be in the form @<comment-file>, which uses the contents of text-file <comment-file> as the comment.

-e <eid>

The element-ID of the element to be undefuncted. If you use this option, you can specify only one element. If you also specify a <list-file> or <element-list>, it is ignored.

This option is useful if multiple elements with the same name have been created, and then *defunct*’ed. Use the element-ID to restore one of the “older” elements; using the name restores the most-recently-*defunct*’ed one.

-l <list-file>

Process the elements listed in <list-file>. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any <element-list> specified is silently ignored.

<element-list>

One or more element names, separated by whitespace. If you also specify a <list-file> using the -l option, this <element-list> is silently ignored.

undefunct

Examples

Restore two previously-*defunct*'ed files:

```
> accurev undefunct old_commands.c old_base.h
```

See Also

add, defunct, promote

unlock

unlock a dynamic stream to enable promotions

Usage

```
accurev unlock [ -kf | -kt ] <stream>
```

Description

The ***unlock*** command removes a lock from a dynamic stream. If you don't specify a ***-k*** option, AccuRev removes the "all" lock.

Options

-kf ("from" lock) Re-enable promotions *from* the stream and purging versions in the stream.

-kt ("to" lock) Re-enable promotions *to* the stream.
You cannot use both ***-kf*** and ***-kt*** in the same command.

Examples

Re-enable promotions to stream **tulip_dvt**:

```
> accurev unlock -kt tulip_dvt
```

See Also

add, lock, promote

update incorporate other people's changes into your workspace

Usage

```
accurev update [ -r <reftree> ] [ -i ] [ -m ] [ -fx ] [ -O ]
               [ [ -t <transaction-number> ] | [ -s <stream-name> -t <transaction-
               range> ] ]
```

Description

The **update** command copies versions from your workspace's backing stream into your workspace. This has the effect of incorporating other people's changes, which they have promoted to the backing stream, into your workspace. It can also be used, with the **-s** and **-t** options, to inspect a stream for changes that occurred within a range of specified transactions; you might want to use the **-s** and **-t** options to monitor a stream that you use for builds, for example.

update replaces only those files in your workspace that are stale. (Use **update -i** to list stale files.) It does not replace files that you have modified, whether you have saved them with **keep** or not. (But **update** can attempt to perform merges on some modified files — see [Handling of \(modified\) Files / Merge on Update](#) below.)

Timestamps on Updated Files

By default, each file copied into your workspace by this command has its timestamp set to the current time. If environment variable ACCUREV_USE_MOD_TIME is set, the timestamp is set to the time the version was originally created. (Note: that's the time the version was created in some user's workspace stream, not the time it was subsequently promoted to the backing stream.) Setting this environment variable may defeat an optimization in the **stat** command. See [Optimized Search for Modified Files](#) on page 247.

Update Level

update causes a new value to be recorded for the update level workspace parameter.

When an **update** command completes, the workspace is fully up-to-date. That is, your workspace has incorporated changes that were recorded in the depot as transactions, up to and including the most recent transaction, say #4167. In this case, transaction #4167 is said to be the update level of your workspace. Subsequent changes, made by you and by other users, might cause the depot's most recent transaction to increase to #4328, but your workspace's update level remains at #4167 until you perform another **update**.

A workspace's update level is displayed by the **show wspaces** command:

```
> accurev show wspaces
...
gizmo_dvt_john    C:/wks/gizmo/dvt_john    moped    6 56 47 1 0
...
```

In this example, the workspace's update level is transaction #47. (The “target level” is transaction #56, indicating that the most recent update of this workspace was interrupted before it could be completed.)

See also [Specifying Ignore Patterns for External Objects](#) on page 7 in *AccuRev Technical Notes*.

update

Handling of (modified) Files / Merge on Update

If you have changed a file in your workspace that is due to be updated, and the file is not yet a member of the workspace's default group, the file has **(modified)(overlap)** status. If there are one or more such files in your workspace, AccuRev's default is to cancel the *update* command:

```
Some of the elements in your workspace are overlap/modified
and are not in your default group.
```

This default protects your data: a simple update would overwrite your changes to the file, which haven't been preserved anywhere in the AccuRev repository.

You have several alternatives for handling this situation:

- *anchor* or *keep* those files, then *update* the workspace. The file(s) that you've processed with *anchor* or *keep* won't be updated.
- *purge* those files, then *update* the workspace. The file(s) that you've processed with *purge* will be updated.
- You can use the *-m* ("merge automatically") option. An *update -m* command succeeds only if each file with **(modified)(overlap)** status can be merged automatically — that is, non-interactively — with the backing-stream version. If this condition is satisfied, the *update* proceeds and each such file is handled as follows:
 - AccuRev performs the automatic merge, replacing the file in your workspace with the merged version. This functionality parallels that of the CVS version-control system.
 - No *keep* or *merge* is recorded in the AccuRev database.
 - The backing-stream version becomes the predecessor of the file in your workspace (because the workspace's update level has changed).

If you have changed a file in your workspace that is *not* due to be updated, and the file is not yet a member of the workspace's default group, the file has **(modified)** status — but not **(overlap)** or **(member)**. In this case, *update* simply bypasses that file (whether or not you use *-m*). After the update finishes, all such files retain their **(modified)** status in the workspace.

Update and Replication

The *update* operation works as follows when you execute it on a client that uses a replica server:

1. An implicit *replica sync* command is performed, copying database transactions from the master repository to the replica repository. This brings the replica database completely up to date.
2. A *stat* operation is performed on the replica server, to determine the state of the workspace stream and its backing stream.
3. Data files representing new versions of elements are copied from the file storage area in the master repository to the file storage area in the replica repository.
4. Data files are copied from the replica repository to your workspace tree. In addition to the files retrieved from the master repository in the preceding step, this can include files that have already been "downloaded" to the replica repository through other users' commands.
5. On both the replica server and the master server, the transaction level of the workspace is set to the most recent transaction (or to the transaction specified with *update -t*).

update

Updating Symbolic Links

On Windows systems, a symbolic link to a directory is implemented as a junction point at the file system level. Accordingly, such links are not supported for FAT/FAT32 file systems. When updating a workspace located in a non-supported file system, an existing symbolic link to a directory generates an error (and *update* creates an empty, standard directory at the link location):

```
Re-linking "dir01sub03.mylink" - failed
```

Updating Reference Trees

Use *update -r* to update a reference tree. To keep a reference tree as up-to-date as possible, call *update* from a job-scheduling program, such as **cron** (UNIX/Linux) or **at** (Windows). You can also use AccuRev's **server-post-promote-trig** trigger to update reference trees automatically. See [Using a Trigger to Maintain a Reference Tree](#) on page 31 in *AccuRev Technical Notes*.

Implementing Partial Workspace Updates

update always processes all the elements in a workspace; there is no option to specify a subset of elements to be updated. You can accomplish this, however, using a series of *merge* commands. See [Using Merges to Implement a Partial Update of Your Workspace](#) in the *merge* reference page.

Options

- i** Show which files would be updated, but don't perform the update.
- m** Enable the "merge on update" capability. See [Handling of \(modified\) Files / Merge on Update](#) above.
- r <reftree>**
Specify a reference tree to be updated. (Default: update the current workspace.)
- s <stream-name> -t <transaction-range >**
Specify a stream and range of transactions you want to inspect for changes. When you use **-s**, you must use **-t** to specify the range of transactions (**-t <#>-<#>**).
- t <transaction-number >**
Update the workspace to the specified transaction. For example, if you specify `update -t 40` and the current transaction level is #45, versions from transactions #41 through #45 are not copied to the workspace.

Note that the transaction number you specify must be greater than the workspace update level. For example, if the current workspace update level is #40, you can specify a transaction number of #41 or more, up to the current transaction level in the depot.

You can see the current workspace update level in these places:
 - In the AccuRev GUI - Select **View > Workspaces** from the main menu. In the Workspaces tab, select Show Details (in the tab's lower right corner). The current update level for all workspaces in the depot is displayed in the Update Level column.
 - In the CLI - execute the *show wspaces* command.
When the **-s** option is used, **-t** is used to specify a range of transactions for the specified stream in the format **-t <#>-<#>**, or **-t 41-45**, for example.
- fx** Display the results in XML format.

update

- O** Override the optimized the search for modified files. When **-O** is used, *all* files are considered, even those whose modification times have *not* changed since the last time the workspace was updated or searched for modified files. Having to check all files, regardless of modification time, slows *update* performance.

Examples

List the files that would be copied into the workspace by an *update* command:

```
> accurev update -i
```

Copy recently created versions into a particular reference tree:

```
> accurev update -r /usr/alpha_test/gizmo4572
```

See Also

co, incl, keep, pop, purge

[The Update Algorithm](#) on page 26 of *AccuRev Technical Notes*

[What's the Difference Between Populate and Update?](#) on page 19 of *AccuRev Technical Notes*

[Specifying Ignore Patterns for External Objects](#) on page 7 of *AccuRev Technical Notes*

upgrade_client

upgrade the AccuRev Client

Usage

```
accurev upgrade_client [ -f ] [ -s ]
```

Description

The *upgrade_client* command compares the AccuRev Server protocol with the AccuRev Client protocol. If these protocols are:

- Different, AccuRev downloads and executes the AccuRev Client installation package that your AccuRev administrator has placed on the AccuRev Server.
- The same, AccuRev displays a message indicating that the AccuRev Client is up to date and that there is no reason to upgrade.

Note that a *protocol* is not the same as a *version*. For example, the AccuRev Server and client might have a different version even if they have the same protocol.

Options

- f Force AccuRev to download and execute the AccuRev client installation package without first checking the AccuRev Server and Client protocols. When you force an upgrade of the AccuRev Client, it is possible, though unlikely, that you might be installing an older version of the AccuRev Client than the one you are currently running. This can happen if your AccuRev administrator has not placed the newer AccuRev Client installation package on the AccuRev Server.
- s After downloading the AccuRev Client installation package, execute the installation "silently". When AccuRev performs a "silent" installation, it uses settings from the existing AccuRev Client installation (AccuRev Server host name and port, for example) to provide information required to perform the installation. If you want to modify or review settings for the AccuRev Client during an upgrade, do not use the *-s* option.

Note: If you are using the *upgrade_client* command to upgrade build machines at your site, you must turn off user access control on those machines. If you do not, the upgrade installation will require you to respond to prompts approving the (unsigned) AccuRev installer.

Examples

Check the AccuRev Server and client protocols; download and execute the client installation package from the AccuRev Server only if necessary:

```
> accurev upgrade_client
```

Download and execute the client installation package from the AccuRev Server without comparing the AccuRev Server and Client protocols:

```
> accurev upgrade_client -f
```

Download and execute the client installation package from the AccuRev Server without first comparing the AccuRev Server and client protocols; use existing AccuRev Client settings to perform a "silent" upgrade:

```
> accurev upgrade_client -f -s
```

verinfo display version info of unarchived elements

Usage

```
verinfo [-n <count>] [-fx] <depot-name>
```

Description

The *verinfo* command displays the version information of unarchived elements in a depot, sorted by size.

Note: The size is the sum of all unarchived versions of the element kept and does not represent the size of the individual file. In addition, elements on Windows machines have the CR (Carriage Return) stripped from LR (Line Return)/CR and may actually be smaller than the actual file.

Options

-fx Display the results in XML format.

-n <count> Display the first <count> elements.

Examples

Display the names of all unarchived elements in a depot:

```
> accurev verinfo depot512
```

Display the top 10 names of all unarchived elements in a depot:

```
> accurev verinfo depot512 -n 10
```

Display the names of all unarchived elements in a depot in XML format:

```
> accurev verinfo -fx depot512
```

wip report work-in-progress for workspaces backed by a stream

Usage

```
accurev wip -s <stream> [ -fx ] [ -a | -d | -u ]
    [ -l <list-file> | <element-list> ]
```

Description

The *wip* (“work-in-progress”) command lists the current activity in all the workspaces that are backed by a particular stream (or a particular set of streams).

Note: Pass-through streams “don’t count”. That is, *wip* considers a workspace that is a child of a pass-through stream to be backed by the closest stream or snapshot in the backing chain that is not a pass-through stream.

A file or directory element is considered to be active in a workspace if you have processed it with any of the following commands (and have not yet either *promoted* or *purged* the change):

- *keep* (including *keeps* performed during *merge* or *patch* commands)
- *anchor*, *co*, *revert*
- *move*
- *defunct*, *undefunct*

Options

-s <stream>

Display the work-in-progress in the workspaces backed by the specified stream. You must specify a stream using *-s*; there is no default.

-a, -d, -u

By default, *wip* considers only the workspaces based on the stream specified with *-s*. These three options expand the set of streams whose workspaces are considered.

- *-a*: all streams
- *-d*: includes all streams that are below the stream specified with *-s*.
- *-u*: includes streams that are in the backing chain of the stream specified with *-s*.

-fx Display the results in XML format.

-l <list-file>

Process the elements listed in <*list-file*>. This must be a text file, with one element name per line. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. Wildcards are not expanded. There is no provision for comment lines in the file.

If you use this option, any <*element-list*> specified is silently ignored.

wip

<element-list>

One or more element names, separated by whitespace. If you also specify a <list-file> using the `-l` option, this <element-list> is silently ignored.

Examples

List all development activity taking place in workspaces backed by stream **tulip_dvt**:

```
> accurev wip -s tulip_dvt
.\src\base.h
tulip_dvt_john
.\tools\perl\findtags.pl
tulip_dvt_mary
.\doc\chap02.doc
tulip_dvt2_john
.\tools\cmdshell\start.sh
tulip_dvt2_john
.\tools\perl_work
tulip_dvt2_john
.\tools\perl_work\reporter.pl
tulip_dvt2_john
```

List activity for files **brass.c** and **brass.h** in the workspaces backed by stream **kestrel_devel**:

```
> accurev wip -s kestrel_devel \.\src\brass.c \.\src\brass.h
.\src\brass.c
kestrel_dvt_mary
.\src\brass.c
kestrel_dvt_john
.\src\brass.h
kestrel_dvt_john
```

See Also

excl, files, incl, stat

Techniques for Selecting Elements on page 10

xml submit a request to the AccuRev Server in the form of an XML message

Usage

```
accurev xml -l <XML-file>
```

Description

The *xml* command submits a request to the AccuRev Server process. It provides a non-interactive general-purpose command dispatcher, supplementing the standard set of CLI commands.

For more information, see [AccuWork™ Command-Line Interface](#) on page 271.

Options

-l <XML-file>

Specifies the location of the XML-format message (document), containing the request for the AccuRev Server.

Example

You can use the *xml* command to query the history of a specific issue:

1. Create an XML file as follows:

```
<historyIssue issueDB = "depot_name">
  <issueNum fid="1">issue_number</issueNum>
</historyIssue>
```

2. Replace *depot_name* with the depot in which the issue is stored.
3. Replace *issue_number* with the issue you want to query.
4. Save the file (as *history.xml*, for example).
5. Run the following command:

```
accurev xml -l history.xml > hist.out
```

ENV_VARS

AccuRev Environment Variables

Usage

```
bash shell:          <AC_ENV_VAR>=<value>
Windows command line:  set <AC_ENV_VAR>=<value>
```

Description

This Help page provides a list of AccuRev environment variables. (There is no “ENV_VARS” command.)

See [AccuRev User Preferences](#) on page 6 of the *CLI User’s Guide* for information about using these settings in the **preferences.xml** file.

Options

ACCUREV_COMMAND_LOGFILE

The *accurev* CLI tool checks whether this variable is defined and contains the pathname of a text file on the client machine. If the pathname is valid, a timestamped copy of the complete command line is appended to the log file when the command begins execution; another timestamped entry is written to the log file when the command completes. (The results of the command are not written to the log file.)

ACCUREV_CONSOLE

(*used by AccuRev Server only*) If this variable is set to 1, the AccuRev Server sends console messages to STDOUT in addition to the **acsrvr.log** file. (Alternative: start the AccuRev Server with the command-line argument **console**.)

ACCUREV_DIFF_FLAGS

diff command: the command-line options to be passed to the file-comparison program. See also [AC_DIFF_CLI](#).

ACCUREV_GUI_INSTANCE

The value of this environment variable dictates the placement of the GUI splash and login screens when running multiple GUIs. Valid values take the format *<m>:<n>*, where *<m>* is the (1-based) number of this particular GUI instance and *<n>* is the total number of GUI instances you intend to run on the client machine. The GUIs can display up to 4 tiled splash and login screens and the value for *<m>* maps to the output display quadrants as follows:

```
 1|2
 3|4
```

Before running a given GUI instance, use this environment variable to specify the desired location. If, for example, you intend to have four GUIs running and you want the next GUI instance to appear in the upper-right quadrant of the screen (quadrant 2), specify the value of this environment variable as **2:4**. Reset the value of this environment variable to the location you want before running the next instance of the GUI. If you run more than four GUI instances, the extra instances start to overlay the first four locations.

ACCUREV_HOME

The full pathname of the parent directory of the **.accurev** subdirectory. Setting this variable overrides AccuRev’s automatic determination of this subdirectory’s location.

ENV_VARS

ACCUREV_SERVER

Dictates the AccuRev Server that the AccuRev GUI interacts with when the GUI is started. Valid values take the format `<server name>:<port>`, for example `bluto:5050` or `cor-e6410:5050`.

If the `ACCUREV_SERVER` environment variable is specified, AccuRev checks to see if the user is already logged in to that server. If the user is logged in, that session is continued and all CLI commands are directed to that server but the `acclient.cnf` file, which contains an ordered list of AccuRev Servers, is not modified. If the user does not have a current session, he or she is prompted to log in. The AccuRev Server specified in the environment variable is preselected in the Log In to AccuRev Server dialog box and is appended to the `acclient.cnf` file if it is not already there.

If the `ACCUREV_SERVER` environment variable is not specified, the user is prompted to log in to the first server listed in the `acclient.cnf` file unless he or she has a current session.

ACCUREV_TOPDIR

Set by the `start` command to the pathname at which the specified workspace or reference tree is located.

ACCUREV_USE_MOD_TIME

If this variable is set to 1, the AccuRev commands `co`, `pop`, `purge`, `revert`, and `update` preserve timestamps when copying versions from the repository into a workspace.

ACCUREV_WATCHDOG_FAST_FAIL_DISABLE

If this variable is set to 1, the AccuRev Server Watchdog process never stops trying to restart the AccuRev Server process, no matter how many Server failures it detects. By default, the Watchdog stops trying after detecting 5 failures in a 3-minute timespan.

ACCUREV_WSPACE

Set by the `start` command to the name of the specified workspace or reference tree.

AC_BROWSER

The full pathname of the Web browser to be used for displaying the help screens of the AccuRev GUI context-sensitive help system.

AC_DIFF_CLI

`diff` command: the file-comparison program to run when this command is invoked. Default: the `acdiff` program in the AccuRev `bin` directory.

AC_EDITOR_CLI

The text editor invoked when the user must enter a comment (e.g. with the `keep` command). This preference has higher precedence than `EDITOR`. If no preference is set, a system-dependent editor is invoked.

AC_EDITOR_GUI

The text editor invoked by the AccuRev GUI File Browser tool when the user invokes the `Edit` command on a text-file element. This preference has higher precedence than `EDITOR`. If no preference is set, a system-dependent editor is invoked.

AC_KEEP_TMP_FILES

Set to `true` to log XML commands sent via `accurev.xml`. Each logged command is saved in a file in a temporary directory under the user's home directory.

ENV_VARS

AC_MERGE_CLI

merge command: the text-file merge program to run when this command is invoked. Default: the *acdiff3* program in the AccuRev **bin** directory.

AC_OLD_UPDATE_PREVIEW

Specifies that the *File > Update Preview* menu option in the AccuRev GUI displays the old (pre-4.9) textual information, rather than the **Update Preview** filter display.

AC_SYNC

Applies only if the user issues an *accurev* command that includes the *-t* option along with a time specification (not a transaction number) *and* there is a timewarp (discrepancy) of more than 5 seconds between the client and server machines.

If not set or set to **ERROR**, a runtime error occurs and AccuRev displays a message that includes the timewarp details.

If set to **WARN**, no error occurs, but a warning that includes the timewarp details is displayed.

If set to **IGNORE**, no error occurs and no warning is displayed.

Note: the values for AC_SYNC are case-insensitive.

EDITOR

The text editor invoked when the user must enter a comment (e.g. with the *keep* command). This preference has lower precedence than [AC_EDITOR_CLI](#). If no preference is set, a system-dependent editor is invoked.

HOME

The directory where the AccuRev configuration subdirectory, **.accurev**, is located. But this can be overridden by an [ACCUREV_HOME](#) specification.

HOMEDRIVE, HOMEPATH

The pathname formed by concatenating of these two preferences specifies the directory where the AccuRev configuration subdirectory, **.accurev**, is located. But this can be overridden by an [ACCUREV_HOME](#) specification.

PATH

Your operating system search path.

SHELL

Your default operating system command processor.

TEMP, TMP

Locations where temporary files can be created.

TZ

Used in rendering the system time as a human-readable string, and reported (if set) by the *info* command.

3. AccuWork™ Command-Line Interface

This chapter describes aspects of the command-line interface (CLI) relevant to the AccuWork issue management system.

Overview

The AccuWork CLI is implemented through a single command, *accurev xml*. The *xml* command is a non-interactive general-purpose command dispatcher; it reads a specified file to determine the AccuRev operation — in this case, an AccuWork command — to be invoked. For example:

```
accurev xml -l mycmd.xml          (“dash-ell” not “dash-one”)
```

Here, the *xml* command’s input comes from a file, *mycmd.xml*, which must contain an XML document. (The filename is irrelevant, and need not have a *.xml* suffix.) The XML document might contain this specification of an AccuWork query:

```
<queryIssue
  issueDB="UserReportedBugs"
  expandUsers="true">
21 == "rel2.0"
</queryIssue>
```

This example specifies the command, “find all issue records in depot **UserReportedBugs** whose value in field #21 (the **targetRelease** field) is the string **rel2.0**”. The results of an *xml* command are sent to standard output, also in the form of an XML document. For example, this query might retrieve two issue records, producing this output:

```
<issues>
  <issue>
    <issueNum
      fid="1">2</issueNum>
    <transNum
      fid="2">3</transNum>
    <targetRelease
      fid="21">rel2.0</targetRelease>
    <type
      fid="7">defect</type>

    ... additional fields ...

    <platform
      fid="12">All</platform>
  </issue>
  <issue>
```

```

<issueNum
  fid="1">3</issueNum>
<transNum
  fid="2">26</transNum>
<targetRelease
  fid="21">rel2.0</targetRelease>
<type
  fid="7">enhancement</type>

... additional fields ...

  <platform
    fid="12">Linux</platform>
</issue>
</issues>

```

This output provides the correspondence between field-ID numbers (e.g. `fid="21"`) and field-names (e.g. `targetRelease`). This correspondence is important, since you must specify a query using field-IDs, not field-names (e.g. `21 == "rel2.0"`, not `targetRelease == "rel2.0"`).

AccuWork CLI Operations

You can perform the following AccuWork operations through the command-line interface:

- Query issues in a depot (`<queryIssue>` document) — Retrieve the contents of all issue records in a particular depot that match a specified query.
- Create a new issue record (`<newIssue>` document) — Enter a single new issue record in a particular depot.
- Modify an existing issue record (`<modifyIssue>` document) — Change the contents of a single issue record that already exists in a particular depot.
- Add or modify an entry in an issue record's change package (`<cpkadd>` document).
- Remove an entry from an issue record's change package (`<cpkremove>` document).
- List the entries in an issue record's change package (`<cpkdescribe>` document).
- Display history in terms of change packages (`<cpkhist>` document) — List transactions and the change packages that they modified.
- Create or remove a Duplicate relationship between two issue records (`<relateIssue>` or `<unrelateIssue>` document).
- Report issue relationships (`<listRelatedIssues>` document) — List records that are related, through the Duplicate relationship, to a particular issue record.
- Compare the change packages in two streams, snapshots, or workspaces (`<issuediff>` document).
- Display the issues whose change packages are in a particular stream, snapshot, or workspace (`<issuelist>` document).

The sections below provide guidelines for performing each of these operations. But there's an important prerequisite step to perform first.

Determining the Field-ID / Field-Name Correspondence

In the AccuWork CLI, you identify a field by its field-ID, not by its field-name. Thus, before doing any real AccuWork CLI work, you must determine the correspondence between field-IDs and field-names for issues in your depot. This information is stored in the schema configuration file on the AccuRev Server host:

```
<AccuRev-inst-dir>/storage/depots/<depot-name>/dispatch/config/schema.xml
```

You can retrieve the contents of the **schema.xml** file from an AccuRev client machine with this command:

```
accurev getconfig -p <depot-name> -r schema.xml
```

Extract the **name=** and **fid=** text lines from this data, and store them for future reference. You'll need to refer to this information often as you work with the AccuWork CLI. Let's call this extracted data the field-ID definitions.

For example, the field-ID definitions for the default schema look like this:

```
name="issueNum"
fid="1">
name="transNum"
fid="2">
name="status"
fid="3">
name="shortDescription"
fid="4">
name="state"
fid="5">
name="productType"
fid="6">
name="type"
fid="7">
name="severity"
fid="8">
name="priority"
fid="9">
name="submittedBy"
fid="10">
name="dateSubmitted"
fid="11">
name="platform"
fid="12">
...
```

This data shows that field **submittedBy** has field-ID **10**, field **productType** has field-ID **6**, etc.

Note: All examples in the remainder of this document will use the field-ID/field-name correspondence in the above example.

The contents of these XML elements are the field values for issue record #1. A couple of them, **submittedBy** and **dateSubmitted**, have values that might be a bit surprising — numbers instead of strings. We'll discuss these kinds of values in section [Selecting Issue Records with a Query](#) below.

Selecting Issue Records with a Query

The simplest kind of query selects one or more issue records by comparing the value of one field with a literal value (a constant) — for example, “is the value of the `issueNum` field equal to 415?” or “does the `shortDescription` field contain the string ‘busted’?”.

For such simple queries, you can adapt the one we used in section [Determining the Field-ID / Field-Name Correspondence](#) above. This query finds all issue records whose `productType` value is **Frammis**.

```
<queryIssue issueDB="XXXXX">
  6 == "Frammis"
</queryIssue>
```

The output of a query is an XML document whose top-level `<issues>` element contains zero or more `<issue>` sub-elements:

```
>>> accurev xml -l query-filename
<issues>
  <issue>
    ... individual field-name elements ...
  </issue>
  <issue>
    ... individual field-name elements ...
  </issue>
  ...
</issues>
```

Where’s the Change Package?

You may notice that in the output of a query, the `<issue>` subelements do not include the issue records’ change package data. Use the `<cpkdescribe>` query to retrieve an issue record’s change package data. See [Listing the Contents of a Change Package](#) on page 282.

More Complex Queries

You can compose and run arbitrarily complex queries. If the query goes just a bit beyond the simplest, you can probably compose it manually. For example, this query finds all issue records whose `productType` value is **Frammis** or **Widget**:

```
<queryIssue issueDB = "dpt38" useAltQuery = "false">
  <OR>
    <condition>6 == "Frammis"</condition>
    <condition>6 == "Widget"</condition>
  </OR>
</queryIssue>
```

With more complex queries, be sure to include the `useAltQuery = "false"` attribute in the `<queryIssue>` start-tag.

But to minimize the chance of getting lost in the syntax of more complex queries, we strongly recommend that you compose the complex query graphically, then “export” the query to a text file:

1. In the AccuRev GUI, enter the command *Issues > Queries* to enter the Query Editor.
2. Create a new query, give it a name, and specify the query logic.
3. Click the *Save All Queries* button.
4. Place an XML-format dump of *all* your queries in a text file:

```
accurev getconfig -p depot-name -u user-name -r query.xml > myqueries.txt
```

This *getconfig* command retrieves the contents of the configuration file that stores your queries:

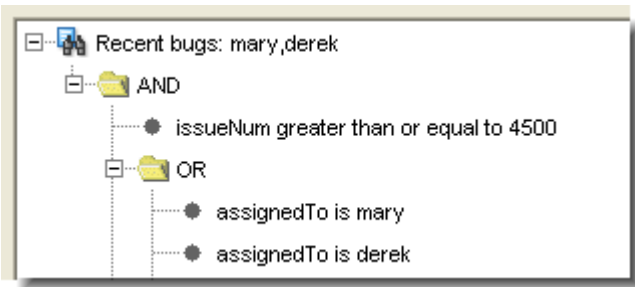
```
AccuRev-install-dir/storage/depots/depot-name/dispatch/config/user/user-name/query.xml
```

5. Use a text editor to extract the XML `<queryIssue>` element that defines the query. (Don't extract the entire `<query>` element, which includes the query's name and output field definitions.)
6. Store the `<queryIssue>` element code in a separate file, say `framemis_or_widget.xml`.

You can now invoke the query with the CLI:

```
accurev xml -l framemis_or_widget.xml
```

Example: This GUI-composed query ...



... is represented by this XML code:

```
<queryIssue issueDB = "dpt38" useAltQuery = "false">
  <AND>

    <condition>
      1 >= &quot;4500&quot;;
    </condition>

    <OR>

      <condition>
        14 == &quot;2&quot;;
      </condition>

      <condition>
        14 == &quot;24&quot;;
      </condition>
    </OR>
  </AND>
</queryIssue>
```

```
</AND>
</queryIssue>
```

This code is perfectly good XML, even though it's not "pretty-printed". Also note the use of the XML entity reference **"**; which is equivalent to a double-quote character.

Special Field Types

Each field in an AccuWork issue has a field type: Text, Choose, List, etc. For a field of type User (such as the **submittedBy** field in the example in section [Determining the Field-ID / Field-Name Correspondence](#) above), a query defaults to reporting users by their integer user-IDs, rather than by their usernames (principal-names):

```
...
<submittedBy
  fid="10">40</submittedBy>
...
```

In this case, you could use the output of the *accurev show users* command to determine that user **john** has user-ID **40**. Alternatively, you can modify the query to set the attribute `expandUsers` in the `<queryIssue>` start-tag:

```
<queryIssue issueDB = "dpt38"
  useAltQuery = "false"
  expandUsers = "true">
  ...
```

Setting `expandUsers` causes the query to report the values of User fields with usernames instead of user-IDs:

```
...
<submittedBy
  fid="10">john</submittedBy>
...
```

For a field of type Timestamp (such as the *dateSubmitted* field in the example in section [Determining the Field-ID / Field-Name Correspondence](#) above), a query always reports the timestamp as a large integer, representing the number of seconds since Jan 1, 1970 UTC:

```
...
<dateSubmitted
  fid="11">1083606273</dateSubmitted>
...
```

(This is the standard UNIX/Linux timestamp scheme.) You can use Perl to convert the integer into a human-readable string:

```
>>> perl -e "print scalar localtime(1083606273)"
Mon May  3 13:44:33 2004
```

Creating a New Issue Record

To create a new issue record in a particular depot, execute the command

```
accurev xml -l my_datafile
```


... where **my_datafile** contains an XML document in this format:

```
<newIssue issueDB="XXXXX">
  <issue>
    ... individual field-value specifications ...
  </issue>
</newIssue>
```

As always, replace **XXXXX** with the name of the depot that the issues are stored in. For the individual field-value specifications, adapt the output of a query that retrieves a single issue record. The complete contents of **my_datafile** might be:

```
<newIssue issueDB="UserReportedBugs">
  <issue>
    <type
      fid="7">defect</type>
    <submittedBy
      fid="10">5</submittedBy>
    <foundInRelease
      fid="20">rel2.0</foundInRelease>
    <productType
      fid="6">Widget</productType>
    <shortDescription
      fid="4">Names are sometimes trunca</shortDescription>
    <dateSubmitted
      fid="11">1083606275</dateSubmitted>  </issue>
  </newIssue>
```

Some DOs and DON'Ts:

- You must specify the value of a User field with a user-ID (**5**), not a username (**derek**).
- You must specify the value of a Timestamp field with a number-of-seconds integer, not a string. You can use the **timelocal()** function in the Perl module **Time::Local** to generate these integers:

```
use Time::Local;
$sec = 35;    # range = 0 .. 59
$min = 22;    # range = 0 .. 59
$hr  = 14;    # range = 0 .. 23
$dte  = 8;    # range = 1 .. 31
$moth = 5;    # range = 0 .. 11    (January is the zero'th month!)
$yr   = 2004; # play it safe: use a 4-digit number
$numseconds = timelocal($sec, $min, $hr, $dte, $moth, $yr);
print $numseconds, "\n";
```

- Don't include specifications for the **issueNum** and **transNum** fields. AccuWork assigns these values automatically.
- The field initialization and validation code defined in the schema will not be executed when the issue record is created. It's up to you to specify the appropriate values for the appropriate fields.

The validations *will* be invoked when the issue record is subsequently opened in the AccuRev GUI. In particular, you can create an issue record with a List field whose value is not in the field's list of

possible values. But when the AccuRev GUI opens the issue record, it replaces the bogus value with `<none selected>`.

- Be sure that the top-level and second-level XML elements are named `<newIssue>` and `<issue>`. The names for the individual-field elements are irrelevant — only the **fid** attributes count. The following specifications are equivalent:

```
<status fid="20">New</status>
<myfield fid="20">rel2.0</myfield>
```

When you submit the `<newIssue>` data structure to the AccuWork CLI, it creates the record and reports the new record's contents. This report includes the automatically assigned **issueNum** and **transNum** values:

```
>>> accurev xml -l my_datafile
<issue>
  <issueNum
    fid="1">11</issueNum>
  <transNum
    fid="2">154</transNum>
  <type
    fid="7">defect</type>
  <submittedBy
    fid="10">24</submittedBy>
  <foundInRelease
    fid="20">rel2.0</foundInRelease>
  <productType
    fid="6">Frammis</productType>
  <shortDescription
    fid="4">Refuses to fram</shortDescription>
  <dateSubmitted
    fid="11">1062787292</dateSubmitted>
</issue>
```

See also [Using 'modifyIssue' to Create New Issue Records](#) on page 280 below.

Modifying an Existing Issue Record

Use the following procedure to modify an existing issue record:

1. Create a query to select the desired issue record, as described in [Selecting Issue Records with a Query](#) on page 274. For example, to select issue record #472 from the **Problems** depot, create this XML document:

```
<queryIssue issueDB="Problems">
  1 == "472"
</queryIssue>
```

2. Run the query, storing the results in a text file:

```
accurev xml -l myquery.xml > issue472.xml
```

3. Edit the text file, making these changes:

- Change the top-level XML `<issues>` start-tag to `<modifyIssue>`.

- Include the **issueDB** attribute in the start-tag:
`<modifyIssue issueDB = "Problems">`
- At the end of the file, change the `</issues>` end-tag to `</modifyIssue>`.
- Remove the entire `<transNum>` element.
- Change the values of one or more existing fields.
- If you wish, add new field values, using the discussion in [Creating a New Issue Record](#) on page 276 as a guide.

Important! Starting in Version 4.6, the XML output file from Step 2 uses UTF-8 encoding, but the input file in Step 4 is interpreted as US-ASCII. Any non-ASCII characters in the file must be coded in decimal entity notation — for example, ` `.

4. Submit the edited text file:

```
accurev xml -l issue472.xml
```

When you submit the `<modifyIssue>` data structure to the AccuWork CLI, it modifies the specified issue record. As when you create a new issue record with the AccuWork CLI, field validations are not applied.

To verify the new record's contents, submit the original query again:

```
accurev xml -l myquery.xml
```

Using the `replaceAll=` Attribute

By default, AccuRev resets any unspecified fields in the `<modifyIssue>` text file to their default values. For example, if your original text file included a `<platform>` element specified like this:

```
<platform
  fid="12">All</platform>
```

but this element is omitted from the text file that you submit using the `accurev xml -l` command, the value for the **Platform** field for that issue record is reset to `<none selected>`.

You can use the `replaceAll=` attribute of the `<modifyIssue>` element to change how AccuRev updates issue records specified in the `<modifyIssue>` text file as follows:

```
<modifyIssue issueDB = "Problems" replaceAll="false">
```

If you set `replaceAll="false"`, AccuRev updates fields using the values specified in the `<modifyIssue>` text file; any fields that are not specified in the file retain their existing values. Returning to the previous example, if you set `replaceAll="false"` and omitted the `<platform>` element from the `<modifyIssue>` text file, its value would remain *All*.

```
<modifyIssue issueDB = "Problems" replaceAll="true">
```

This is the default behavior described above. You do not have to explicitly set `replaceAll="true"`.

Bulk-updating Issue Records

In addition to changing field values of individual issue records, you can use *bulk update* to update multiple issues with the same changes to one or more fields. For example, you can use bulk update to change the value of the **Status** field to *Scheduled* for the issue records you specify.

To use bulk update, you use the XML to specify:

- The records you want to update
- The fields and values that you want to change

Returning to the previous example, if you wanted to change the value of the **Status** fields for issues 72 and 73 to *Scheduled*, and add a comment to the **Description** field, your XML might look like this:

```
<modifyIssue issueDB="accurev" replaceAll="false">
  <issues>
    <issue>
      <issueNum fid="1">72</issuenum>
    </issue>
    <issue>
      <issueNum fid="1">73</issuenum>
    </issue>
  </issues>
  <values>
    <status fid="3">Scheduled</status>
    <description fid="16">Needs to be fixed ASAP</shortDescription>
  </values>
</modifyIssue>
```

You can use bulk update to modify most fields in this fashion. See the discussion in [Creating a New Issue Record](#) on page 276 for more information.

Using 'modifyIssue' to Create New Issue Records

Instead of using a **<modifyIssue>** document to change an existing issue record, you can use it to create a new issue record. This is useful for copying issue records from one depot to another. For example, you might use a **<queryIssue>** document, as described earlier, to retrieve the contents of an issue record from depot **Problems**, in the form of an **<issues>** document. As described in Step 3 above, when you change the **<issues>** tag to a **<modifyIssue>** tag, specify a different depot:

```
<modifyIssue issueDB = "Problems_Public">
```

In this example, the issue record will effectively be copied from the **Problems** depot to the **Problems_Public** depot. Make sure the target depot exists and has the same schema as the source depot.

The **<modifyIssue>** technique is also useful for making copies of the issue records that act as change packages, and for replicating issue records between different AccuRev sites.

Note that a new issue record created with **<newIssue>** always gets assigned the next available issue number; by contrast, a new issue record created with **<modifyIssue>** gets the issue number specified by the **<issueNum fid="1">** subelement:

```
<issueNum
  fid="1">4197</issueNum>
```

An issue record with this number must not already exist in the target depot, but there is no other restriction. It's perfectly OK for most issue numbers to be unallocated.

Interface to the Change Package Facility

AccuWork issue records are used to implement the change package facility. The set of changes in a change package is indicated by a set of “Versions”, listed on the Changes subtab of an issue record, each with a corresponding “Basis Version”:

Element	Version	Basis Version
\doc\chap01.doc	8/1	9/2
\doc\chap04.doc	9/3	9/2
\src\brass32.c	9/30	9/27
\src\brass32.h	9/9	9/3

Each Version / Basis Version pair defines a set of changes to the element: the changes made since the Basis Version was created, up to and including the Version. The change package consists of such Version / Basis Version “patches” for any number of elements.

Various user commands and triggers maintain the contents of a change package: adding new versions and removing existing versions. There are also commands for comparing the contents of a change package to the contents of a stream, enabling you to easily answer the question, “Have all the changes made for Task A been propagated to Stream B?”

The following sections describe the CLI to the change package facility.

Adding Entries to a Change Package

The following XML document requests the adding of two entries to the change package of issue record #433: for the element with element-ID 3, record the series of versions between Basis Version 4/3 and Version 4/6; for the element with element-ID 7, record the series of versions between Basis Version 4/2 and Version 4/9.

```
<acRequest>
  <cpkadd>
    <user>john</user>
    <depot>etna</depot>
    <stream1>etna_dvt</stream1>
    <issues>
      <issue>
        <issueNum>433</issueNum>
      </issue>
    </issues>
    <elements>
      <element
        id="3"
        real_version="4/6" basis_version="4/3">
      </element>
      <element
        id="7"
        real_version="4/9" basis_version="4/2">
      </element>
    </elements>
  </cpkadd>
</acRequest>
```

```

        </element>
      </elements>
    </issue>
  </issues>
</cpkadd>
</acRequest>

```

Removing Entries from a Change Package

The following XML document requests the removal of the change-package entry for the element with element-ID 3 from issue record #433.

```

<acRequest>
  <cpkremove>
    <user>john</user>
    <depot>etna</depot>
    <stream1>etna_dvt</stream1>
    <issues>
      <issue>
        <issueNum>433</issueNum>
        <elements>
          <element
            id="3"
            real_version="4/6">
          </element>
        </elements>
      </issue>
    </issues>
  </cpkremove>
</acRequest>

```

Listing the Contents of a Change Package

The following XML document requests the listing of the change packages of issue records #433 and #512.

```

<acRequest>
  <cpkdescribe>
    <user>john</user>
    <depot>etna</depot>
    <stream1>etna_dvt</stream1>
    <issues>
      <issueNum>433</issueNum>
      <issueNum>512</issueNum>
    </issues>
  </cpkdescribe>
</acRequest>

```

Note that a `<cpkdescribe>` query is the only way to retrieve an issue record's change package data. A `<queryIssue>` query retrieves all the other fields, but not the change package data.

Listing Transactions that Affected Change Packages

The following XML document requests the listing of certain transactions in the range 488–569, including the numbers of the change packages (issue records) modified by those transactions.

```
<acRequest>
  <cpkhist>
    <user>john</user>
    <depot>etna</depot>
    <transaction1>569</transaction1>
    <transaction2>488</transaction2>
  </cpkhist>
</acRequest>
```

The transactions listed include explicit change-package requests (kind=**dispatch**, operation=**cpkadd** or **cpkremove**). Also included are **promote** transactions that triggered **cpkadd** operations.

Creating a Relationship between Two Issue Records

The following XML document creates a Duplicate (type=1) relationship between issue records #337 (the parent) and #198 (the child):

```
<relateIssue issueDB="UserReportedBugs">
  <relationship
    type="1"
    issueNum1="337"
    issueNum2="198">
  </relationship>
</relateIssue>
```

Removing a Relationship between Two Issue Records

The following XML document removes a Duplicate (type=1) relationship between issue records #337 (the parent) and #198 (the child):

```
<unrelateIssue issueDB="UserReportedBugs">
  <relationship
    type="1"
    issueNum1="337"
    issueNum2="198">
  </relationship>
</unrelateIssue>
```

Listing Issue Record Relationships

Following is an XML template for reporting ...

- the relationships in which a specified issue record is the parent, or
- the relationships in which a specified issue record is the child, or

- both of the above

```
<listRelatedIssues issueDB="depot-name">
  <relationship
    type="type-number"
    issueNum="issue-number"
    relationship="kind">
  </relationship>
</listRelatedIssues>
```

The placeholders in this template are:

depot-name

The name of the depot in which the issue resides.

type-number

- 1 (Duplicate)
- 3 (Sub Task)
- 4 (Dependency)
- 5 (Tracking)

issue-number

The integer number of the issue record whose relationships are to be listed.

kind

child, to list the issue records that have *issue-number* as a child.

parent, to list the issue records that have *issue-number* as a parent.

all, to combine both the above listings.

Determining Which Change Packages Are Included in a Stream

The following XML document requests a *Show Issues* command on stream **etna_dvt**: list the change packages (issue records) whose entire set of changes have been propagated to stream **etna_dvt**.

```
<acRequest>
  <issuelist show_active= "false">
    <user>jjp</user>
    <depot>etna</depot>
    <stream1>etna_dvt</stream1>
    <issues>
    </issues>
  </issuelist>
</acRequest>
```

Changing the attribute to `show_active = "true"` would restrict the listing to change packages that have at least one version that is currently active in stream **etna_dvt**. (Removing the attribute would have the same effect, since the “show_active” attribute defaults to “true”.)

The following variant (note the **boldface** code) restricts the listing to a particular set of active change packages (issue records): in effect, this allows you to pose the question, “Have all the changes in #167 and #332 already been propagated to stream **etna_dvt**?”.

```
<acRequest>
  <issuelist>
    <user>jjp</user>
    <depot>etna</depot>
    <stream1>etna_dvt</stream1>
    <issues>
      <issueNum>167</issueNum>
      <issueNum>332</issueNum>
    </issues>
  </issuelist>
</acRequest>
```

Other attributes for the `<issuelist>` element include:

- `show="[complete | incomplete]"` -- show only complete or incomplete issues from the specified stream.
- `show_unaffiliatedchanges="[true | false]"` -- show changes that are not associated with any issue in the specified stream.

Comparing Two Streams in Terms of Change Packages

The following XML document requests a *Show Issues Difference* command, showing which change packages (issue records) in stream **brass_dvt** have not had their changes propagated to stream **brass_tst**.

```
<acRequest>
  <issuediff>
    <user>john</user>
    <depot>brass</depot>
    <stream1>brass_dvt</stream1>
    <stream2>brass_tst</stream2>
  </issuediff>
</acRequest>
```

The following variant (note the **boldface** code) restricts the listing to a particular set of change packages (issue records): in effect, this allows you to pose the question, “Are #167 and #332 among the change packages that have not yet been propagated from stream **brass_dvt** to stream **brass_tst**?”.

```
<acRequest>
  <issuediff>
    <user>john</user>
    <depot>brass</depot>
    <stream1>brass_dvt</stream1>
    <stream2>brass_tst</stream2>
    <issues>
      <issueNum>167</issueNum>
      <issueNum>332</issueNum>
    </issues>
  </issuediff>
</acRequest>
```

```
</issuediff>  
</acRequest>
```

Other attributes for the `<issuediff>` element include:

- `show="[complete | incomplete]"` -- include only complete or incomplete issues from the diffed streams.
- `show_unaffiliatedchanges="[true | false]"` -- show changes that are not associated with any issue in either stream.

4. XML Output Reference

This chapter provides reference information for the XML output of some AccuRev commands.

How Command Information is Organized

Commands are listed in alphabetical order. Each command includes one or more samples of the XML output that results from executing it. Where possible, an explanation of the information contained in an XML tag is displayed on the same line. These comments are displayed in a different font, like *XML declaration* in the following example:

```
<?xml version="1.0" encoding="utf-8"?> XML declaration
```

For formatting considerations, when the description of a tag's values requires additional space, the description includes *See below for details*.

Finally, tag descriptions with an asterisk (*) indicate that the tag is included in the XML output only if the value for that tag has meaningful data.

hist Command

Displays the part or all of the transaction history of one or more elements. See [hist](#) on page 120 for more information.

```
<?xml version="1.0" encoding="utf-8"?>           XML declaration
<AcResponse      root element of the XML output
  Command="hist"  the name of the AccuRev command
  TaskId="1037"> the ID of the task associated with the command
  <element        a list of one or more elements returned for certain transactions. See below for details.
    name="/outline.txt" the element location and name
    id="2">          the element ID
    <transaction  a list of transactions for this element. See below for details
      id="3"       the transaction ID
      type="add"   the transaction type. See below for details
      time="1407174305" the transaction time in Epoch time
      user="dfoster"> the name of the user who executed the transaction
      streamName="phoenix_n" stream name in which the transaction occurred (promote only)*
      streamNumber="1" the number of the stream (promote only)*
      fromStreamName="phoenix_n_dfoster" the name of the source stream (promote only)*
      fromStreamNumber="2"> the number of the source stream (promote only)*
      cpk="add"> type of change package action (if applicable). values are "add" and "remove".*
      <comment>new file</comment> comment associated with the transaction*
      <version    a list of versions for this transaction. See below for details.
        status="(member)" element status (appears with -fs)*
        virtual="2/1" element virtual version (stream ID/version in stream)*
        real="2/1" element real version (stream ID/version in stream)*
```

```

virtualNamedVersion="phoenix_n_dfoster/1"      See below for details*
realNamedVersion="phoenix_n_dfoster/1"        See below for details*
elem_type="text"    element type: dir, text, binary, ptext, elink, slink. See below for details
dir="no"    whether or not the element is a directory; not displayed for dispatch transactions*
ancestor="2/2"    ancestor stream ID and version; not displayed for purge or eacl
                  commands*
ancestorNamedVersion="wfl_update_testuser/2"    ancestor stream ID and
                                                  version; not displayed for
                                                  purge or eacl commands*
merged_against="2/2"    merged against stream ID and version; not displayed for
                        purge or eacl commands*
mergedAgainstNamedVersion="f1_update_testuser/2"    merged against
                                                    stream name and
                                                    version; not displayed
                                                    for purge or eacl
                                                    commands*

new_revert="yes"    internal use only. indicates whether or not the reverse patch
                    algorithm was used*

target_eid="2"    for elinks, the ID of the element to which this element is linked*
target_name="no"    for elinks, the ID of the element to which this element is linked*
sz="17">    version element size, in bytes; displayed only if version attribute dir="no"*
</version>
</transaction>
</element>
<streams>    a list of all the streams and workspaces whose IDs are included in the <version>
              information for an element

  <stream
    id="2"    stream ID
    name="phoenix_n_dfoster"    stream name
    type="workspace"/>    stream type. values: normal, workspace, snapshot, gated,
                          staging, passthrough

  </streams>
</AcResponse>

```

Additional Information

This section provides additional information for some of the child elements of `<AcResponse>` in the XML output for the **hist** command.

<element>

The **hist** command returns a list of one or more elements only for the following transactions types:

add, co, defcomp, defunct, demote_from, demote_to, dispatch, eacl, keep, move, promote, purge, undefunct

If there is no history associated with the specified element, AccuRev returns:

```
<Message>No history corresponding to selection.</Message>
```

<element>
<transaction>

If the transaction type is not one supported by the **hist** command, AccuRev returns:

```
<Message>No such transaction: -3.</Message>
```

<element>
<transaction>
<stream>

The **<stream>** child element of the **<transaction>** element is displayed in the XML output for **mkstream**, **chstream**, and **defcomp** transactions.

- If the transaction was **mkstream** or **defcomp**, the XML output for **<transaction>** includes the **<stream>** child element with the following attributes. For example:

```
<transaction  
  <stream  
    name="dev"           stream name  
    streamNumber="3"     stream number  
    depotName="phoenix_n" depot in which stream was created  
    type="normal"       stream type. values: normal, workspace, snapshot, gated, staging,  
                        passthrough  
    basis="phoenix_n"   name of the stream's basis stream  
    basisStreamNumber="1"/>  
  </stream>
```

- If the transaction was **chstream**, the following attributes are added to the **<stream>** child element. For example:

```
<transaction  
  <stream           same attributes as those returned for mkstream and defcomp transactions, plus:  
    prev_name="dev" the stream's previous name  
    time="0"        for time-based streams, basis time in Epoch time  
    prevTime="0"    for re-set time-based streams, previous basis time in Epoch time  
    prevBasis="phoenix_n" name of the stream's previous basis stream  
    prevBasisStreamNumber="1"> number of the stream's previous basis stream  
    prevHidden="true"> if the stream was previously hidden (values are "true" and "false")  
  </stream>
```

Note: The **type=** attribute for the **<transaction>** element returns **"chstream"** for **chws** commands executed on workspaces.

```

<element>
  <transaction>
    <stream>
      <wspace>

```

If the **mkstream** or **chstream** command was executed against a workspace, the `<wspace>` child element appears under `<stream>`. For example:

```

<transaction
  <stream
    ...
    <wspace
      Storage="C:/feenix_n/workspaces/phoenix_n"    workspace storage location
      Host="con-davidf-e6410"                       the name of the computer that hosts the file system
                                                    on which the workspace is located
      Target_trans="11"                             the most recent transaction at the time of the workspace's
                                                    most recent update
      fileModTime="1407248340"                       the time of the last update or stat in Epoch time
      EOL="0"                                         end-of-line character: 0=OS default; 1=force UNIX; 2=force Windows
      Type="1"/>    workspace type: 1=full workspace; 9=exclusive locking; 17=anchor required
    </stream>

```

```

<transaction>
  <version>

```

A list of one or more versions in the transaction.

- The `elem_type` attribute displays the element type. Values include `dir`, `text`, `binary`, `ptext`, `elink`, and `slink`. It is not displayed for the **dispatch** transactions. For example:

```

<version
  ...
  elem_type="text"    values include dir, text, binary, ptext, elink, slink
  ...

```

- If the transaction was **move**, the XML output for `<version>` includes the following (for example):

```

<move
  dest=".\\subdir1\\subdir11\\foo.c"    destination and element name of the move
  source=".\\subdir1\\subdir11\\foobar.c"/>    source and element name of the move

```

- If you use `v` option with `-fx`, the XML output for `<version>` includes, for example:

```

<version
  path=".\\outline.txt"    element name and location in workspace
  eid="2"                element ID
  mtime="1407524402"    version modification time in Epoch time; displayed only if dir="no"
  cksum="2514403315"    version element checksum; displayed only if dir="no"
  <eacl                element access control list

```

```

    action="set"                eacl action; values are "add", "remove", and "set"
    <ace                        a list of all ACEs for this element
      principal="all"          values are "all", "<user_name>", and "<group_name>"
      privilege="all"/>       values are "all", "none", and "clear"
  </eacl>

```

```

<transaction>
  <version>
    <patchSegment>

```

Patch segments are displayed in the XML output only if the transaction type is not **purge** and the element has no ancestor. There can be multiple `<segment>` child elements. For example:

```

<version
  ...
  <patchSegments>           displays patch segments
    <segment                displays patch segment details
      head_stream="5"       segment head stream ID
      head_stream_name="whmp2_testuser" segment head stream name
      head_version="2"      segment head stream version
      basis_stream="5"      segment basis stream ID
      basis_stream_name="whmp2_testuser" segment basis stream name
      basis_version="1"     segment basis stream version
      is_tip_version="yes"/> segment is a tip version ("yes" or "no")
    </patchSegments>

```

```

<transaction>
  <version>
    <revertSegment>

```

Revert segments are displayed in the XML output only if the transaction type is not **purge** and the element has no ancestor. There can be multiple `<segment>` child elements. For example:

```

<version
  ...
  <revertSegments>        revert segments
    <segment              displays revert segment details
      head_stream="5"     segment head stream ID
      head_stream_name="whmp2_testuser" segment head stream name
      head_version="2"    segment head stream version
      basis_stream="5"    segment basis stream ID
      basis_stream_name="whmp2_testuser" segment basis stream name
      basis_version="1"   segment basis stream version
      is_tip_version="yes"/> segment is a tip version ("yes" or "no")
    </revertSegments>

```

<transaction> <comp_rule>

The <comp_rule> child element appears for the <transaction> element if one of the following include/exclude rules was applied to the element: clear, excl, incl, incldo. Displays attributes for cross-linked elements if applicable.

```
<comp_rule
  kind="incl"           type of rule: clear, excl, incl, incldo
  xlinkStreamNum="4"   the ID of the stream to which the element is cross-linked
  xlinkStreamName="hist2" the name of the stream to which the element is cross-linked
  prevXlinkStreamNum="2" if include rule was reset, the ID of the previously
                        cross-linked stream
  prevXlinkStreamName="hist2a"/> if include rule was reset, the ID of the previously
                        cross-linked stream
```

In addition, when an include/exclude rule is in force, the value of the `type=` attribute for the <transaction> element displays "defcomp" (short for "define component"), as shown here:

```
<transaction
  id="3"
  type="defcomp"       "defcomp" displayed when clear, excl, incl, or incldo rules are in force
  time="1407174305"   time of the transaction in Epoch time
  ...
```

<element> <transaction> <issues>

The <issues> child element appears for <transaction> if you are using change packages (CPK) to promote elements. The <issues> child element can have multiple <issue> child elements.

```
<issues>           displays change package issues
<issue>           displays issue details
  <issueNum        the AccuWork issue number; attributes used for third-party ITS, if any,
  thirdPartyName="MyJiraKey" the name of the third-party ITS key field, if any,
  thirdPartyValue="J1003">4</issueNum> the attribute is the third-party key value
  <elements>      elements in the change package; there can be multiple <element> child elements
  <element         element details
    eid="5"        element ID
    member="1"     element member number
    head_version="5/4" element head version
    basis_version="5/3" element basis version
    name="Access denied"/> change package name; displays "Access denied"
                        if the user executing hist command lacks
                        sufficient rights

  </elements>
```



```
</issue>
</issues>
```

Isrules Command

Displays the include/exclude rules for a workspace or a stream. See [lsrules](#) on page 151 for more information.

```
<?xml version="1.0" encoding="utf-8"?>           XML declaration
<AcResponse                                     root element of the XML output
  Command="lscomp"                             internal name for the set (kind) of rules returned by the lsrules command
  TaskId="1292">                               the ID of the task associated with the command
  <element
    kind="incl"                                the type of rule associated with the element: clear, incl, incls, excl
    elemType="dir"                             the element type
    dir="yes"                                  whether or not the element is a directory
    location=".\"                               the element's location
    setInStream="phoenix_n"                   the name of the stream in which the rule was set
    options="1"/>                             indicates whether rules were created using AccuRev 4.5.3 or later
                                                0= created using a version of AccuRev earlier than version 4.5.3
                                                1= created using AccuRev 4.5.3 or later
  </element>
</AcResponse>
```

