The TimeSafe® Configuration Management System

AccuRev User's Guide

(Command Line Interface)

Version 3.8

August, 2005

AccuRev User's Guide (CLI)
August, 2005

Table of Contents

AccuRev User's Guide (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 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*. This is important, because AccuRev works differently than CM systems with a branches-and-labels architecture.

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) 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**.)

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 is a directory hierarchy, containing a set of version-controlled files and directories. Depots are completely discrete from one another: they 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 nested within one another.

Example: suppose a depot contains this directory hierarchy:

```
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.

```
H:
cd \Projects\Colorwheel\derek\doc
... enter accurev commands ...
```

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
```

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
```

Note the special prefix — `\.\` for Windows, `/./` for Unix — 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.

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/repository 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. For more information on these administrative files, see *Using Multiple AccuRev Servers* on page 43 in *AccuRev Technical Notes*.

Depot Setup

We'll assume that a depot (data repository) has already been set up for your use. Instructions for creating a depot and populating it with files to be version-controlled are included in the several "Quick Evaluation" chapters in *AccuRev Technical Notes*.

Getting Work Done with the CLI

AccuRev does not force you to proceed with your software development work in any particular way. We're here to help you, not put you in a straitjacket! 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 in Touch With Your Past*
- *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_jjp

- **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)
  /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. A sparse workspace (**–ks**) contains copies of certain elements only: the elements you load into the workspace with the **pop** command.

  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. To create a workspace in which you must use the **co** or **anchor** command to "check out" a file before editing it, specify the **–ka** option. The **–ke** option goes even further, providing exclusive file locking: after you use **co** or **anchor** to start working on a file, users in other workspaces 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 (**–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 4), 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 above), 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 10.)

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, which describes 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 in Touch With Your Past

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, immutable 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 (**create**, even though the command-name is **add**), when you rename it or move it to a different directory (**move**), when you incorporate someone else's changes into your work (**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; 2003/10/28 14:34:44 ; user: derek
  dst : \.\src\White.java
  src : \.\src\W.java
  version 6/6 (6/6)
  ancestor: (6/5)

transaction 43019; keep; 2003/10/28 14:27:40 ; user: derek
  # improve error message
```

```
version 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
brass_dvt_mary
   \.\tools\perl\findtags.pl
   \.\tools\perl\reporter.pl
brass_dvt_jjp
   \.\doc\chap01.doc
   \.\doc\chap04.doc
   \.\doc\procfiles.py
   \.\tools\perl\reporter.pl
brass_dvt_derek
   \.\doc\procfiles.py
   \.\src\brass.c
   \.\src\brass.h
```

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 commands). It’s even careful not to overwrite files with changes that you haven’t yet preserved with **keep**.

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 19.

Managing a Depot's Stream Hierarchy

Creating Data Structures	
Command	**Description**
mkdepot	create a new depot
mkref	create a new reference tree
mksnap	create a new static stream
mkstream	create a new dynamic stream
mkws	create a user workspace
mklinks	populate a workspace with links (Unix-specific)
Maintaining Data Structures	
Command	**Description**
chdepot	rename a depot
chref	change the name and/or definition of a reference tree
chslice	change the location of a slice
chstream	change the name and/or definition of a stream
chuser	rename a user
chws	change the name and/or definition of a workspace
reactivate	restore a reference tree, stream, user, or workspace to active service
remove	remove a reference tree, stream, user, or workspace from active service

The data repository managed by AccuRev 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 stream. You can change the specifications of an existing stream: its name, its location in the stream hierarchy, and its time basis (**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**).

Managing and Creating New Versions of Files

Command	Description
add	add a new element to a depot
anchor	add an element to the default group of a workspace
chmod	change the access mode of an element (Unix-specific)
co	(check out) add an element to the default group of a workspace
defunct	remove an element from a stream
keep	create a new version of an element
merge	merge changes from another stream into the current version of an element
move, *mv*	move or rename elements
patch	incorporate the changes from one other version into the current version
pop	copy files into a workspace or reference tree
promote	propagate a version from one stream to another stream
purge	undo all of a workspace’s changes to an element
revert	“undo” a promote transaction
start	create a command shell in a workspace or reference tree
touch	update the timestamp of a file
undefunct	restore a previously removed element to a stream

Command	Description
update	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, 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 5 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 **keep** command (see *Checkpointing — Saving Private Versions* on page 6) 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 **merge** command (see *Incorporating Other Users' Work into Your Workspace* on page 10) creates and **keep**s 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 **defunct** command removes an element from your workspace. The **undefunct** command restores a previously **defunct**ed element to your workspace.

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 **revert** command performs one or more **co** commands, the effect of which is to "roll back" the effects of a specified **promote** command, making your workspace look as if you'd purged your work instead of promoting it. (See **purge** description below.)

The **promote** and **purge** commands transition an element from active status to passive status. These two commands are opposites. **promote** (see *Making Your Changes Public* on page 7) takes a private version — created by **keep** or **move** or **defunct**, etc. — and makes it public by sending it to the backing 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 10) 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. 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-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
annotate	indicate the origin of each line of a text file
cat	get the contents of a version of an element
diff	compare two versions of an element
dir	list elements in a stream
files	show the status of elements
hist	show the transaction history of elements or an entire depot
info	show basic information about the current session
mergelist	determine which versions need to be promoted between streams
name	list the name of the element with the specified element-ID
patchlist	list versions that need to be patched into the workspace's version
show	list all depots, streams, workspaces, or slices
stat	show the status of elements
translist	list transactions containing versions that need to be promoted
type	alias for **cat** command

Command	Description
wip	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 8) list the status of elements in a workspace, or in a stream. (So does the **dir** command, which is now deprecated.) 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 **wip** command (see *Tracking Other Users' Work* on page 10) 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 Facility

Command	Description
incl	include elements in a workspace or stream
incldo	include just a directory, not its contents, in a workspace or stream
excl	exclude elements from a workspace or stream

Starting in Version 3.5, AccuRev's include/exclude facility replaces the sparse workspace facility. These commands make it easy to include just the files you need in a workspace or stream.

Administration

Command	Description
backup	prepare the AccuRev repository for data backup
mktrig	activate a trigger in a depot

Command	Description
ping	test client-server communications link
rmtrig	deactivate a trigger in a depot
synctime	synchronize system clock on client computer to server computer

AccuRev has remarkably little administrative overhead. The **backup** command prepares the repository for a live backup — there's no need to stop the AccuRev Server process. The **mktrig** and **rmtrig** commands maintain the repository's triggers, which control users' ability to make changes to depots.

The **ping** command tests the client-server communication channel. The **synctime** adjust a client machine's system clock to match that of the AccuRev server machine.

Managing Users and Security

Command	Description
addmember	add a user to a group
chgroup	rename a group
chpasswd	change the password of a user
chuser	rename a user
ismember	does a particular user belong to a particular group?
lock	lock a dynamic stream against promotions
lsacl	show access control list entries
mkgroup	create a new group of users
mkuser	register a new username
rmmember	remove a user from a group
setacl	create an access control list entry
setlocalpasswd	create an authn file on the local machine
unlock	unlock a dynamic stream that was locked against promotions

AccuRev maintains 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**, **chpasswd**, and **setlocalpasswd** commands change the name and password settings for an existing user-ID.

Similarly, the **mkgroup** and **chgroup** commands maintain the repository's set of 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.

AccuRev Command Line Reference

This document 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) 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 document 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 recognizes the following commands:

Command	Description
add	add a new element to a depot
addmember	add a user to a group
anchor	add an element to the default group of a workspace
annotate	indicate the origin of each line of a text file
archive	prepare to transfer version container files to offline storage
backup	prepare the AccuRev repository for data backup
cat	get the contents of a version of an element
chdepot	change the properties of a depot
chgroup	rename a group
chmod	change the access mode of an element (Unix-specific)
chpasswd	change the password of a user
chref	change the name and/or definition of a reference tree
chslice	change the location of a slice
chstream	change the name and/or definition of a stream
chuser	rename or relicense a user
chws	change the name and/or definition of a workspace
clear	remove an include/exclude rule

co	(check out) add an element to the default group of a workspace
defunct	remove an element from a workspace
diff	compare two versions of an element
dir	list elements in a stream
excl	exclude elements from a workspace or stream
files	show the status of elements
hist	show the transaction history of elements or an entire depot
info	show basic information about the current session
incl	include elements in a workspace or stream
incldo	include just a directory, not its contents, in a workspace or stream
ismember	does a particular user belong to a particular group?
keep	create a new version of an element
lock	lock a dynamic stream against promotions
lsacl	show access control list entries
lsrules	show the include/exclude rules for a workspace or stream
merge	combine changes from another stream into the current version of an element
mergelist	determine which versions need to be promoted between streams
mkdepot	create a new depot
mkgroup	create a new group of users
mklinks	populate a workspace with links (Unix-specific)
mkref	create a new reference tree
mksnap	create a new static stream
mkstream	create a new dynamic stream
mktrig	activate a trigger in a depot
mkuser	register a new username
mkws	create a user workspace
move, *mv*	move or rename elements
name	list the name of the element with the specified element-ID
patch	incorporate a set of changes from a given workspace into the current version

patchlist	list versions that need to be patched into the workspace's version
ping	test client-server communications link
pop	copy files into a workspace or reference tree
promote	propagate a version from one stream to another stream
purge	undo all of a workspace's changes to an element
reactivate	make a reference tree, stream, user, or workspace visible again
reclaim	remove archived version container files from gateway area
remove	hide a reference tree, stream, user, or group
rename	alias for **move** command
replica	synchronize a replica repository
revert	"undo" a promote transaction
rmmember	remove a user from a group
rmtrig	deactivate a trigger in a depot
rmws	hide a workspace
setacl	create an access control list entry
setlocalpasswd	create an authn file on the local machine
show	list all depots, streams, workspaces, or slices
start	create a command shell in a workspace or reference tree
stat	show the status of elements
synctime	synchronize system clock on client computer to server computer
translist	list transactions containing versions that need to be promoted
touch	update the timestamp of a file
type	alias for **cat** command
unarchive	restore version container files that were previously archived
undefunct	restore a previously removed element to a workspace
unlock	unlock a dynamic stream that was locked against promotions
update	incorporate other people's changes into your workspace
wip	report work-in-progress for workspaces backed by a stream

Return Values

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

Command 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 ...
```

All options are single-dash (for example, **–a**), not double-dash ("--absolute"). DOS-style options (**/a** instead of **–a**) are not supported.

Common Command Options

Most **accurev** commands accept options, such as **–a** and **–m**. A few options are recognized by many of the commands, and work identically (or nearly identically) in each command. You can combine these common options in powerful ways, which you might not think of right away.

Not all of the options are compatible with each other and not all of the options are accepted by all commands. Once you learn what each option is used for, using them in practice should quickly become second nature.

The common options are primarily used to select elements to operate on. For instance, you may want to perform a **keep** command on all the elements in a directory, or on just one element, or on all the elements you have modified. You can select exactly the elements you want quickly and simply using command options.

Basic Selection

The most basic way to select elements is to select them individually by listing them on the command line. You can also use wildcard characters, such as **?** and *****.

Many of the selection options work as filters. That is, they narrow down the list of elements based on certain criteria. To keep all of the elements in a directory:

```
accurev keep *
```

To keep just the modified elements you would enter:

```
accurev keep -m *
```

Selecting Everything

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
```

Element Status

Each element has a status associated with it:

Value	Description
backed	The element is the same as the most recent version in the backing stream. Either it has not been changed since the workspace tree was last updated, or you changed the element and promoted the change to the backing stream.
external	The file is located in the workspace tree but has not been **add**ed to the depot (and so, it's not yet an element).
kept	The changes to the element have been put in the workspace stream using **keep**. No changes have been made to the element since that time.
modified	The element has been changed since the last time the workspace tree was updated, or since the last time the file was kept.
member	The element is active in the workspace, and so is included in the workspace's default group. The following commands "activate" an element in the current workspace: **anchor**, **co**, **revert**, **keep**, **move**, **defunct**, **undefunct**.
stale	The element has been changed in the backing stream, but has not been updated in the workspace tree.
overlap	The element has been changed both in the workspace tree and in the backing stream. Before you can **promote** your changes to the backing stream, you must reconcile the differences with the **merge** command.
missing	There should be a version of the element in the workspace, but there isn't. This occurs when you delete a version-controlled file from the workspace, using an operating system command or another application.
stranded	The element is in the default group of the workspace, but has become stranded. This occurs when the workspace no longer has a pathname to the element.

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").

Selecting Elements Based on Their Status

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 26, 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, the active elements are the ones that you've modified and/or kept, but have not yet promoted to the backing stream.
- In a (non-workspace) stream, the active elements are those that have been promoted to the stream "from below", but have not yet been promoted to the parent stream.

When you do a **co** (checkout) operation or a **keep** operation on an element, it is added to the default group (if it is not already there). 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 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
```

Selecting Elements 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 — i.e. 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
```

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 (e.g. 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
- **–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.

Setting the Element Type

AccuRev supports two element types: **text** and **binary**. To override the type that would be selected for an element by default, use the **–E** option. This option is available in the **add** and **keep** commands. To create an element with type **binary**:

```
accurev add -E binary foo.doc
```

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)

You can use a forward slash (/) or a backslash (\ ) on any platform. With Unix 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 — i.e. 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 steam "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.

Specifying the AccuRev Server

For most **accurev** commands, you can specify the AccuRev server/repository to target with 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**. The hostname/port-number argument to this option has the same form as in the **acclient.cnf** file.

This mechanism bypasses the **acclient.cnf** file, though the file must still exist. It does *not* override a specification in the **wspaces** file. (These configuration files are described in *Using Multiple AccuRev Servers* on page 43 of *AccuRev Technical Notes*.)

add — add a new element to a depot

Usage

```
accurev add [ -c <comment> ] [ -E <elem-type> ] [ -x ] [ -R ]
   { -l <list-file> | <element-list> }
```

Description

The **add** command converts one or more existing files and/or directories 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 depot's database for an **add** command, the AccuRev operation is listed as "create", not "add". Transactions are listed by the **hist** command.

add creates 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.

Controlling the Element Type

add guesses an element type for each file element it creates:

- If the file is strict ASCII, it sets the element type to **text**.
- Otherwise, it sets the element type to **binary**.

  Note: Unicode files are considered to be binary, not text.

Text files are stored in the depot with a single NL (or LF) character (hex character code **0A**) at the end of each text line. By default, 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 (**0A**) for a Unix machine, or CR-NL (**0D 0A**) for a Windows machine. You can force a workspace to use a particular line terminator using the **–e** option to **mkws** or **chws**.

You can override **add**'s element type determination with the **–E** option, described below. To intelligently set the element type based on file extensions and/or contents, set a pre-operation trigger with **mktrig pre-create-trig**. AccuRev ships with a sample trigger script, **elem_type.pl**. You can easily customize it to recognize new file extensions.

To see a file element's type, use the **hist** command. To change an element's type, use the **keep –E** command. This change affects future versions only; it does not change the type of any existing version.

Controlling the Unix Access Mode

On a Unix 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.

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 AC_EDITOR_GUI (or named in environment variable EDITOR, or in a system-dependent default editor).

–E <*elem-type*>

Specify the element type: **text** (default) or **binary**. See *Controlling the Element Type* above.

–l <*list-file*>

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

If you use this option, you cannot also specify an <*element-list*>.

<element-list>

One or more filenames, separated by whitespace. If you specify a list of elements on the command line, you cannot also use the **–l** option.

–R Recurse into the directory specified in <*element-list*>, and add all the external files in that directory. (You must also specify **–x**, and <*element-list*> must consist of a single directory name. Use "**.**" to specify the current working directory.)

–x Select all external files and directories in the workspace.

Preferences

The following preferences are implemented through environment variables. Setting of environment variables differs among operating systems and shell programs.

ACCUREV_IGNORE_ELEMS

Takes a space-separated list of filename patterns. Causes **add –x** to ignore external files that match any of the patterns.

Example: in the Unix Bourne shell, have **accurev add –x** ignore temporary files and text-editor backup files:

```
export ACCUREV_IGNORE_ELEMS="*.tmp *.bak"
```

(In general, enclose the pattern list in quotes on Unix systems, but not on Windows systems.) See also *Using the ACCUREV_IGNORE_ELEMS Environment Variable* on page 39 of *AccuRev Technical Notes*.

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

Common Command Options on page 22, **chmod**, **hist**, **keep**, **mktrig**, **promote**

addmember — add a user to a group

Usage

```
accurev addmember <principal-name> <group-name>
```

Description

The **addmember** command adds an existing AccuRev principal-name to an existing AccuRev group.

AccuRev groups implement security, through access control lists (ACLs), and also establish development roles.

Examples

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

```
accurev addmember john_smith eng
```

See Also

mkgroup, **chgroup**, **mkuser**, **rmmember**, **show groups**, **show members**, **setacl**, **lsacl**

anchor — add an element to the default group of a workspace

Usage

```
accurev anchor [ -n ] { -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" 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.

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.

Exclusive File Locking

In a workspace where exclusive file locking is in effect (serial-development mode), or in an anchor-required workspace:

- The files that you are not actively working on are read-only. To work on a file, first enter a **co** or **anchor** command; this makes the file writable.
- (exclusive file locking only) The **co**/**anchor** command places an exclusive file lock on the file, which is honored by other exclusive file locking workspaces based on the same backing stream.

  Within a set of exclusive file locking workspaces, if a file is under active development (**anchor**, **co**, **revert**, **keep**, **move**, **defunct**, **undefunct**) in one workspace, users in the other workspaces cannot modify that file. In a workspace where exclusive file locking is not in effect (parallel-development mode), all files are writable and existing locks don't affect the ability to **co** files. See *Serial Development through Exclusive File Locking* on page 25 of the *AccuRev Concepts Manual*.

Options

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

–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. There is no provision for comment lines in a list-file.

If you use this option, you cannot also specify an <*element-list*>.

<element-list>

One or more element names, separated by whitespace. If you specify a list of elements in this way, you cannot also use the **–l** option.

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**

annotate indicate the origin of each line of a text file

Usage

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

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 date of that transaction.

By default, **annotate** lists the current version of the file in your workspace. You can use the **–v** option to specify any other version.

By default, **annotate** includes all of the annotations described above. You can use the **–f** option to specify certain annotations, and to specify the order in which they appear.

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 26 for a description of the forms that <*ver-spec*> can take.

–f...
: **–ft**: (“transaction”) The transaction in which this line was added to the file, or was most recently modified.
–fu: (“user”) The user who performed that transaction.
–fd: (“date”) The timestamp of that transaction.

You can specify multiple format options. For example, **–fut** annotates the text lines with the user and transaction number, in that order. By default, all annotations are included, as if you’d specified **–ftud**.

Examples

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

```
> accurev annotate -ftu factors.py
      1 jjp            def gcf(big, small):
      1 jjp                  """
      1 jjp                  find the greatest common factor of two numbers
      1 jjp                  """
      1 jjp
      1 jjp                  # special cases
      1 jjp                  if big == small: return big
     21 mary                 # oops, wrong order
      1 jjp                  if big < small:
      1 jjp                         big, small = small, big
      1 jjp
```

```
13 mary                 # reduce, using the classic algorithm
 1 jjp                  while big % small > 0:
 1 jjp                          big, small = small, big % small
 1 jjp
13 mary                 # return greatest common factor
 1 jjp                  return small
 1 jjp
 1 jjp          def lcm(big, small):
 1 jjp                  """
 1 jjp                  find the least common multiple of two numbers
 1 jjp                  """
13 mary                 return big * small / gcf(big,small)
 1 jjp
 1 jjp          def prime_factors(n):
 1 jjp                  """
 1 jjp                  return a list of the prime factors of a number
 1 jjp                  """
21 mary                 factors = []
```

See Also

cat

archive — prepare to transfer version container files to offline storage

Usage

```
accurev archive [ -R ] [ -s <stream> ] [ -t <transaction-range> ]
   [ -c <comment> ] [ -i ] [ -E <element-type> ] <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:

- It focuses on a particular stream. If you don't specify a stream with **–s <*stream*>**, it uses your current workspace's stream.
- It narrows the scope to a particular set of file elements, which you specify as command-line arguments in the **<*element-list*>**. You can include directories in this list; in this case, use the **–R** option to include the recursive contents of those directories.
- By default, all eligible versions of the specified elements in the specified stream are archived. You can use **–t** to limit the set to the versions of those elements created in a specified transaction, or range of transactions:

–t <*number*>	*single transaction*
–t <*number*>–<*number*>	*range of transactions*

- You can also limit the set of versions to those of a particular element type, using the **–E** option.

The **archive** command refuses to archive any version that is currently visible in any stream or snapshot.

Using the **–i** option (in addition to the other options described above) generates an XML-format listing of the desired versions, 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 25 of the *AccuRev Administrator's Guide*.

See Also

reclaim, **unarchive**

backup — prepare the AccuRev repository for data backup

Usage

```
accurev backup mark
```

Description

The **backup** command declares a "checkpoint" of the AccuRev repository. This involves making copies of certain repository files, and takes just a few seconds. After you've declared a checkpoint, you can proceed to make backup copies of all the files in the repository, including the central site slice directory tree and all the individual depot directory trees. 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.

To checkpoint the repository, the **backup** command:

- Copies the database files (**.ndb**) from the site slice directory to a subdirectory named **backup**. It also copies the index file **stream.ndx** to the subdirectory.
- Records the current state of each depot's database files in file **valid_sizes_backup** in the depot directory.

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. See *The 'maintain' Commands Related to 'backup mark'* on page 48 of the *AccuRev Administrator's Manual*.

This procedure returns the repository to its state at the time you "checkpointed" it with the **backup** command.

See Also

Backing Up the Repository on page 3 of the *AccuRev Administrator's Manual*.

cat, type get the contents of a version of an element

Usage

```
accurev {cat|type} [ -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 an element from depot storage. It 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.

By default, **cat** displays the version in your workspace stream. (The contents of this version differ from the contents of the file in your workspace, if you've made changes but haven't yet kept them.) You can specify any version in any stream, using the **–v** option.

To display a file in your workspace, you don't need the **accurev cat** command — just use the command shell's **cat** (Unix) or **type** (Windows) command.

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 26 for a description of the forms that <*ver-spec*> can take.

–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. You cannot specify the element with a simple filename; use its depot-relative pathname (or its element-ID) instead.

–e *<eid>*
: Operate on the 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
```

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 }
```

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.

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* on page 113.

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 completely remove the depot from the AccuRev repository, using the AccuRev administration utility, **maintain**. Here's the procedure:

1. Rename the depot:

   ```
   accurev chdepot -p mercury mercury.RMV
   ```

2. Stop the AccuRev Server process, then invoke the **maintain** command to remove the depot from the repository:

   ```
   maintain rmdepot mercury.RMV
   ```

   This command requires careful confirmation, so that you don't inadvertently remove needed data.

3. Restart the AccuRev Server process, then create a new depot with the original name:

   ```
   accurev mkdepot -p mercury
   ```

Note that after you perform the above procedure, the name **mercury.RMV** remains in the repository, and cannot be reused.

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.

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
```

See Also

mkdepot, **chstream**, **show depots**

chgroup — rename a group

Usage

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

Description

The **chgroup** command changes the name of a user group.

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

mkgroup, **addmember**, **show groups**

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

Usage

```
accurev chmod { ugo+x | ugo-x } <element-list>
```

Description

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

The **chmod** command changes the Unix 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.

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

chpasswd — change the password of a user

Usage

```
accurev chpasswd <principal-name> <new-password>
```

Description

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

Note: you can change only your own password, not the password of another user.

Password Storage and Password-Change Procedure

Your password is stored under your home directory, within subdirectory **.accurev**, in file **authn**. On Windows systems, your home directory is determined by concatenating the values of the environment variables HOMEDRIVE and HOMEPATH. On some versions of Windows, you must set these environment variables yourself; they are not set automatically by the operating system.

The password is stored as plaintext (not encrypted) in the **authn** file. For security reasons, this file must be owned by you and be read-only; all other users and groups must have no access rights to this file.

Setting your password for the first time, with **chpasswd** or with **mkuser**, creates the **authn** file. Changing your password thereafter involves verifying that:

- Your AccuRev username is <*principal-name*>.
- Your **authn** file contains the correct password for <*principal-name*>.

Then, **chpasswd** stores the new password in the AccuRev database and writes the new password to the **authn** file.

> Note: if you use AccuRev on more than one client machine, the several machines may access different **authn** files. If so, be sure to change the contents of **authn** on other client machines, using the **setlocalpasswd** command on each machine.

Examples

Change your password to **janedoe** (your AccuRev username must be **john_smith**):

```
accurev chpasswd john_smith janedoe
```

See Also

setlocalpasswd, **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 5 of the *AccuRev Administrator's Manual*.

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) 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).

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 line terminator, NL), or **w** (use the Windows line terminator, CR-NL).

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 5 of the *AccuRev Administrator's Manual*.

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) or **xcopy** (Windows).

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.

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
```

See Also

mkdepot, **show depots**, **show slices**

chstream — change the change the name and/or definition of a stream

Usage

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

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

Description

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

- the name of the stream
- 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**.

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: e.g. **2001/08/07 20:27:15**
- Time keyword: **now**
- Transaction number as a positive integer: e.g. **146** or **23965**
- Transaction number keyword: **highest**
- Time keyword: **none** (to remove an existing basis time from a stream)

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, **show streams**, **remove stream**

chuser — rename or relicense a user

Usage

```
accurev chuser { -kf | -kd } <principal-name> <new-name>
```

Description

The **chuser** command changes the principal-name of an existing AccuRev user and/or changes the user's AccuRev licensing.

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

Options

–kf ("full") Change the user to be licensed for both configuration management and issue management (Dispatch).

–kd ("Dispatch") Change the user to be licensed for issue management (Dispatch) only.

Examples

Rename a user:

```
accurev chuser john_smith johnny
```

Change a user so that he can use Dispatch only.

```
accurev chuser -kd kevin
```

See Also

mkuser, **show users**, **remove user**

chws — change the 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> ]
   [ -k <kind> ] [ -e <eol-type> ]

accurev chws -w <workspace>
   { -b <new-backing-stream> | -r <new-reftree> }

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

Description

Note: before changing the location of a workspace, consult *A Word of Caution on Windows Zip Utilities* on page 5 of the *AccuRev Administrator's Manual*.

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

- name of the workspace
- kind of workspace
- location of the workspace tree (hostname and/or pathname)
- the stream or reference tree associated with the workspace

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) 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.

<new-name>

Specify a new name for the workspace; this also renames the workspace stream. The name must not start with a digit. **chws** adds the suffix _<*principal-name*> to the name you specify (if you don't include the suffix yourself).

–k <*kind*>

Change the kind of workspace. See the description of **mkws** for details on the specifiers **d** (default storage scheme), **l** (links, Unix-only), **e** (exclusive file locking), and **a** (anchor required).

As of Version 3.5, you cannot change a workspace to kind **s** (sparse). See the descriptions of **incl** and **excl**.

–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 line terminator, NL), or **w** (use the Windows line terminator, CR-NL).

–r <*new-reftree*>

(Unix-only) For a link-based workspace, change the reference tree whose files will be the targets of the links.

Examples

After moving the contents of workspace **amber_dvt_mary** 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
```

See Also

mkref, **mkws**, **show refs**, **show wspaces**, **incl**, **excl**

clear — remove an include/exclude rule

Usage

```
accurev clear [ -s <stream> ] <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.)

Example

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

incl, **excl**, **lsrules**, **update**

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

Usage

```
accurev co [ -n ] [ -R ] [ -v <ver-spec> ]
   { -l <list-file> | <element-list> }

accurev co -t <transaction-number>
```

Description

The **co** 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.

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.

Exclusive File Locking

In a workspace where exclusive file locking is in effect (serial-development mode), or in an anchor-required workspace:

- The files that you are not actively working on are read-only. To work on a file, first enter a **co** or **anchor** command; this makes the file writable.
- (exclusive file locking only) The **co**/**anchor** command places an exclusive file lock on the file, which is honored by other exclusive file locking workspaces based on the same backing stream.

  Within a set of exclusive file locking workspaces, if a file is under active development (**anchor**, **co**, **revert**, **keep**, **move**, **defunct**, **undefunct**) in one workspace, users in the other workspaces cannot modify that file. In a workspace where exclusive file locking is not in effect (parallel-development mode), all files are writable and existing locks don't affect the ability to **co** files. See *Serial Development through Exclusive File Locking* on page 25 of the *AccuRev Concepts Manual*.

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.) Setting this environment variable may defeat an optimization in the **stat** command. See *Optimized Search for Modified Files* on page 155.

Options

–n Select all of the modified elements in the workspace that are not already in the default group.

–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.)

–R Recurse into the directory specified in <*element-list*>, and check out all 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. There is no provision for comment lines in a list-file.

If you use this option, you cannot also specify an <*element-list*>.

<element-list>
One or more element names, separated by whitespace. If you specify a list of elements in this way, you cannot also use the **–l** option.

Using Checkouts or 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 **co –v** commands to do so. For example, if your backing stream is named **gizmo_dvt** and you want to update elements **foo.c**, **bar.c**, and **read.me**, this set of commands accomplishes the "partial update":

```
accurev co -v gizmo_dvt foo.c
accurev co -v gizmo_dvt bar.c
accurev co -v gizmo_dvt read.me
```

(You'll probably want to use a script or a loop-processing command to make this task user-friendly.) If a file is active in your workspace, you can use the **merge** command instead of the **co** command to incorporate changes from the backing stream.

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

Common Command Options on page 22

defunct — remove an element from a workspace

Usage

```
accurev defunct { <element-list> | -l <list-file> | -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) 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 "promote"* below).

Defuncting Directories

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.
- 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.

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.

Propagating the Change with "promote"

After **defunct**ing an element, you use **promote** to propagate this change 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.

Reusing the Name of a Defunct Element

After **defunct**ing an element, you can then create a new file with the same name. The file is an external object — it exists only in the workspace tree, and is unrelated to the defunct element in the workspace stream.

AccuRev does not allow you to **add** the new file to the depot, because this would create a set of twins: two elements at the same pathname in the same stream. (One is the **defunct**ed element, the other is the newly **add**ed element.)

If you promote the **defunct**ed element to the backing stream, that element is removed from the workspace stream. This "frees up" the pathname, making it possible to **add** the new file to the depot.

> Note: If you *do* **add** the new file to the depot, then **promote** the newly created element to the backing stream, a twin situation occurs in the backing stream:

- The newly created element is active in the backing stream, and can be inherited by lower-level streams and workspaces.
- The original element, now **defunct**ed, is also active in the backing stream. Its status includes the **(stranded)** and **(twin)** indicators. Because it is defunct, this element cannot be inherited by lower-level streams and workspaces.

Reactivation with "undefunct"

To bring back a defunct element, use the **undefunct** command. You can do this either before or after you promote the change to the backing stream. (For special cases, see the description of **undefunct**.)

The Past and Future of a Defunct Element

In the future, an element that you have **defunct**ed 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.

Exclusive File Locking

The **defunct** command fails if both of these conditions hold:

- You are in a workspace where exclusive file locking is in effect (serial-development mode).
- Some other workspace based on the same backing stream already has any of the specified files under active development (**anchor**, **co**, **revert**, **keep**, **move**, **defunct**, **undefunct**).

See *Serial Development through Exclusive File Locking* on page 25 of the *AccuRev Concepts Manual*

Options

–e <*eid*>

Operate on the element with the specified element-ID. You can use this option instead of specifying a list of 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. There is no provision for comment lines in a list-file.

If you use this option, you cannot also specify an <*element-list*>.

<element-list>

One or more element names, separated by whitespace. If you specify a list of elements in this way, you cannot also use the **–l** option.

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
```

See Also

hist, **promote**, **stat**, **undefunct**

diff — compare two versions of an element

Usage

Compare file in your workspace with another version:

```
accurev diff [ -b | -v <ver-spec> ] [ -cw ] [ -G ] <element-list>
```

Compare two specified versions:

```
accurev diff -v <ver-spec-1> -V <ver-spec-2> [ -cw ] [ -G ] <element-list>
```

Compare two streams:

```
accurev diff -a -i -v <ver-spec-1> -V <ver-spec-2> [ -cw ]
```

Use filter to determine set of elements to process:

```
accurev diff [ -a -d -k -m -o -p ] [ -i ] [ -cw ] [ <stream-specs> ]
```

The <*stream-specs*> can be:

<none>	Compare the file in your workspace with the version in the workspace's backing stream.
–v <*ver-spec*>	Compare the file in your workspace with the version specified with **–v**.
–v <*ver-spec*> **–V** <*ver-spec*>	Compare the two specified versions.
–v <*ver-spec*> **–b**	Compare the version specified with **–v** with the version in the workspace's backing stream.
–v <*ver-spec*> **-1**	("dash-one") Compare the version specified with **–v** with its immediate predecessor version.
–b –1	("dash-one") Compare the version in the workspace's backing stream with its immediate predecessor version.

Note: at the end of any **diff** command, you can add:

```
-- <diff-utility-flags>
```

These flags are passed through to the program that performs the actual file comparison. (See *Controlling the Comparison Process* below.)

Description

The **diff** command compares two versions of a file element (or a set of file elements). By default, it shows the changes you've made recently to a file — since an **update** of the workspace or a **keep** of the file.

More precisely, the default is to compare the file in your workspace with the version listed by the **stat** command:

- the active version in the workspace stream (if the element is active in your workspace)

- the version inherited from the backing stream or a higher-level stream (if the element is not active in your workspace)

The **–b** option forces a comparison with the backing stream's version, even if the element is active in your workspace. The **–v** option forces a comparison with a specified version.

Finding the Differences between Two Streams

diff –a –i –v –V is a very powerful command for showing the differences between two streams. It lists just what has changed and how it has changed. It indicates the starting version and the ending version and whether elements have been moved, defuncted, or undefuncted.

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.

Options

Specifying the Elements

You must specify at least one element — directly in an element-list, or indirectly with one or more of these element-selection options.

–a Select all elements in the depot.

–d Select only elements in the default group.

–k Select only elements that you have kept but not yet promoted.

–m Select only elements that have been modified.

–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.

Specifying the Versions to Compare

By default, one of the versions to be compared is the file in your workspace. Use one or more of the following options to specify the other version, or to specify both versions

–b Select the version used by the backing stream as one of the versions to be compared.

–v ***<ver-spec>***
Specify a particular historical version (stream-ID/ver-number) or the version currently used by a particular stream (stream-ID).

–V ***<ver-spec2>***
Use this option along with **–v** to specify the second version to be used in the file comparison.

–1 ("one", not "el" — meaning "the one preceding") Determine the second version to be used in the file comparison as follows. Take the first version — the one specified with **–v** or **–b**

— and consider its stream (S) and the transaction (T) that created it. Use as the second version the one that was in stream S as of transaction T–1.

Common cases: If the first version was created in a workspace with **keep**, the **–1** option selects the version on which the first version 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 preceding version created in that stream.

Controlling the Comparison Process

The following options control the way in which the two selected versions of an element are compared as text files.

–c Context diff.

–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.

–w Ignore whitespace in the file comparison.

–G Use the graphical Diff tool to perform the file comparison.

– – *<diff-utility-flags>*

Any items on the command line following a double-dash (--) are passed directly to the program that performs the actual file comparison.

You can use the environment variable ACCUREV_DIFF_FLAGS to hold default flags to pass to the file-comparison program. If you use – – to specify pass-through flags on the command line, this environment variable is ignored.

Preferences

The following preferences are implemented as environment variables.

AC_DIFF_CLI

A command string that **diff** will use to invoke an alternative file-comparison program. (The default file-comparison program is **acdiff** in the AccuRev **bin** directory.) The command string must specify the two versions to be compared with the substitution patterns **%1** and **%2**.

Make sure that the file-comparison program you specify is on your search path.

AC_DIFF_GUI (no longer supported!)

Prior to Version 3.0.2, AccuRev supported a preference AC_DIFF_GUI, which specified a graphical-file-comparison program to be invoked by the **Diff Against ...** GUI commands. As of Version 3.0.2, you must specify such a program in the GUI window, with the command **File > Preferences > Diff**. See *Enabling the Diff and Merge Tools* on page 95 of the *AccuRev User's Guide (GUI)*.

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
```

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 2000)
```

Return Values

diff returns 0 for no differences, 1 for differences, or 2 for an error.

See Also

merge, **mergelist**

dir — list elements in a stream

Usage

```
accurev dir [ -s <stream> ] <element-list>
```

Description

Note: starting with AccuRev Version 3.2, use of the **dir** command is deprecated. Use the **files** command instead.

The **dir** command displays the names of elements that exist in a stream. For each directory element you specify as an argument, **dir** lists the elements contained in the directory, rather than listing the directory itself. The most typical use of **dir** is to list the contents of the current working directory in the workspace stream:

```
% cd /home/jjp/workspaces/gizmo_dvt_jjp/lib
% accurev dir .
/./lib/libcmd      (dir)
/./lib/libgui      (dir)
/./lib/libutil     (dir)
/./lib/libweb      (dir)
```

Workspace Stream vs. Workspace Tree

Comparing the **dir** and **stat** commands highlights the difference between a workspace tree (an ordinary directory tree on your machine's disk drive) and the corresponding workspace stream (a directory tree that exists only in the AccuRev repository database). In sum, the **stat** command lists the actual files and directories in the workspace tree, and the **dir** command lists the version-controlled elements in the workspace stream. In detail:

- **dir** lists only file and directory elements. It ignores external files and directories — e.g. text-editor backup files and temporary directories — which exist in the workspace tree but are not elements. By contrast, the **stat** command does list external files and directories.
- **dir** lists an element in a workspace stream regardless of whether a corresponding file or directory currently exists in the workspace tree. If not, it lists the element with a **(missing)** annotation. An element in the workspace stream can be missing from the workspace tree if:
  - You used operating system commands to delete, rename, or move the item in the workspace tree.
  - The workspace is sparse, and you never loaded that particular element into the workspace with the **pop** command.

  By contrast, when scanning a directory the **stat** command lists only files and directories that currently exist in the workspace tree.

Options

–s <*stream*>

Display elements in the specified stream. Default: display elements in the workspace stream.

<element-list>

One or more element names, separated by whitespace. For a file element, **dir** lists the element itself. For a directory element, **dir** lists the elements cataloged in that directory. Names of external or non-existent files and directories are silently ignored.

Examples

Go to a directory and list the elements it contains:

```
% cd lib
% accurev dir .
\.\lib\libweb\index.htm
\.\lib\libweb\products.htm
\.\lib\libweb\support.htm
\.\lib\libweb\products.gif
\.\lib\libweb\support.gif
\.\lib\libweb\logo.gif
```

Delete a few files from a directory, then list all of the directory's elements:

```
% del lib\libweb\*.gif
% accurev dir lib\libweb
.\lib\libweb\index.htm
.\lib\libweb\products.htm
.\lib\libweb\support.htm
.\lib\libweb\products.gif   (missing)
.\lib\libweb\support.gif   (missing)
.\lib\libweb\logo.gif   (missing)
```

List the elements in a top-level directory, in a sparse workspace that has loaded only one of the subdirectories:

```
% accurev dir lib
\.\lib\libcmd        (dir)  (missing)
\.\lib\libgui        (dir)  (missing)
\.\lib\libutil       (dir)  (missing)
\.\lib\libweb        (dir)
```

See Also

files, **stat**

excl — exclude elements from a workspace or stream

Usage

```
accurev excl [ -s <stream> ] <element>
```

Options

–s <*stream*>
Apply the exclude rule to the specified stream.

Description

Note: the include/exclude facility replaces the sparse workspace facility. As of Version 3.5, you can no longer create a workspace of kind "sparse" (**mkws –ks**).

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.

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

incl, **incldo**, **lsrules**, **clear**, **update**

files — show the status of elements

Usage

```
accurev files [ -s <stream> ] [ -O ] <element-list>
```

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 doesn't consider files whose timestamps precede the workspace's update time (the time that the workspace was most recently updated).

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.

See also *Search Performance Optimizations* on page 59 of the *AccuRev User's Guide (GUI)*.

Using 'update' to Improve 'files' Performance

The timestamp optimization described above produces the greatest performance boost when it enables **files** to ignore a large number of files based on their timestamps. If **files** seems sluggish, try executing an **update** command. If the workspace contains a significant number of files whose timestamps fall between the previous update and the current time, the optimization will enable a subsequent execution of **files** to ignore these files.

Listing of Defuncted / 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_jjp\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**, annotated with the **(defunct)** status flag, until you **promote** it or **purge** it. Promoting the defunct element removes it from the workspace 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 **dir** 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: don't optimize the search for modified files (i.e. don't ignore files whose modification times precede the workspace's update time). The override also enables **files** to report certain elements as "missing" from the workspace. Having to check all files, regardless of modification time, slows **files** performance. See *Optimized Search for Modified Files* above.

<element-list>
: One or more element names, separated by whitespace. 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 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_jjp\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_jjp\2 (2\2) (kept) (member)
.\doc\chap03.doc     brass2\1 (2\1) (backed)
.\doc\chap04.doc     brass2\1 (2\1) (backed)
```

See Also

dir, **stat**

hist — show the transaction history of elements or an entire depot

Usage

```
accurev hist [ -t <transaction-range> ] [ -s <stream> ]
   [ -u <principal-name> ] [ -k <transaction-kind> ] [ -fevstx ]
   { -a | <element-list> | -e <eid> }

accurev hist -p <depot-name> [ -k <transaction-kind> ]
   [ -t <transaction-range> ] [ -fevstx ]
```

Description

The **hist** command displays the 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. You can use this option instead of specifying a list of elements.

–p <*depot-name*>
Specify the depot to use (default: the depot of the current workspace).

–s <*stream*>
Display only the transactions that involved the specified stream.

–f... **–fe**: (“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.
–fv: (“verbose”) For **keep** transactions and **create** transactions (**add** command), include the modification time, checksum, file size, data type (text or binary), and depot-relative pathname.
–fev: Both expanded and verbose.
–fs: (“status”) If an element is in the default group of the stream specified with **–s** (default: workspace stream), annotate it with “**(member)**”. This option is intended to be used in combination with **–t**, when you’re examining a transaction listed by **translist**.

–ft: (“transaction”) Display just the header line, including the transaction number.
–fx: (“XML”) Display results in XML format.

–k <*transaction-kind*>
Display only the transactions of the specified kind (promote, keep, move, etc.).

–u <*principal-name*>
Display only the transactions for the specified AccuRev user.

You can specify a <*transaction-range*> in these ways:

–t <*time-spec*>[.<*count*>]
Display one transaction, or a specified number of transactions up to (i.e. preceding) and including a particular transaction.

A time-spec can be any of the following:

- Time in <YYYY/MM/DD HH:MM:SS> format: e.g. **2001/08/07 20:27:15**
- Time keyword: **now**
- Transaction number as a positive integer: e.g. **146** or **23965**
- Transaction number keyword: **highest**

–t <*time-spec*> – <*time-spec*>[.<*count*>]
Display transactions that took place in the specified interval. The optional dot-suffix truncates the listing after the first <*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.

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”. 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
```

See Also

translist

incl include elements in a workspace or stream

Usage

```
accurev incl [ -s <stream> ] <element>
```

Options

–s <*stream*>

Apply the include rule to the specified stream.

Description

Note: the include/exclude facility replaces the sparse workspace facility. As of Version 3.5, you can no longer create a workspace of kind "sparse" (**mkws –ks**).

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) 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. You may need to use one or more **incldo** commands to satisfy this requirement before you issue the **incl** command.

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.

Examples

Include the **python** subdirectory of the previously excluded **tools** directory in the current workspace:

```
accurev incl \.\tools\python
```

See Also

incldo, **excl**, **lsrules**, **clear**, **update**

incldo — include just a directory, not its contents, in a workspace or stream

Usage

```
accurev incldo [ -s <stream> ] <directory-element>
```

Options

–s <*stream*>

Apply the include-directory-only rule to the specified stream.

Description

Note: the include/exclude facility replaces the sparse workspace facility. As of Version 3.5, you can no longer create a workspace of kind "sparse" (**mkws –ks**).

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 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 incl   tools\scripts\python
```

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 override this exclusion for particular files/directories under it.

Examples

Include just the top-level **readme.txt** file and the **online** subdirectory from the hierarchy **widget\support\documentation\online**:

```
accurev incldo widget
accurev incldo widget\support
accurev incldo widget\support\documentation
accurev incldo widget\support\documentation\online
cd widget\support\documentation\online
accurev incl readme.txt
```

See Also

incl, **excl**, **lsrules**, **clear**, **update**

info — show basic information about the current session

Usage

```
accurev info [ -v ]
```

Description

The **info** command displays the basic characteristics of the AccuRev user environment:

```
Principal:      jjp
Host:           jake
client_time:    2001/09/19 15:13:03 Eastern Daylight Time (1000926783)
```

If your current directory is within a workspace, **info** also displays information regarding the workspace and its associated streams:

```
Depot:          accurev
Stream:         jake5_jjp
Basis:          accurev_int
Top:            d:/accurev_wks/jake5
```

Options

–v Display additional information: the version numbers of the AccuRev client and server programs.

Examples

Get your bearings:

```
> accurev info

Principal:      jjp
Host:           biped
server_name:    localhost
port:           5051
ACCUREV_BIN:    C:/Program Files/AccuRev/bin
client_time:    2004/01/19 12:20:29 Eastern Standard Time (1074532829)
server_time:    2004/01/19 12:20:32 Eastern Standard Time (1074532832)
Depot:          accurev
ws/ref:         accurev_docwk_jjp
Basis:          accurev_int
Top:            c:/accurev_depot
```

See Also

start

ismember — does a particular user belong to a particular group?

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 **jjp** a member of AccuRev group **ctdvt**?

```
> accurev ismember jjp ctdvt
1
```

See Also

addmember, **rmmember**, **show**

keep — create a new version of an element

Usage

```
accurev keep [ -c <comment> ] [ -dmn ] [ -E <element-type> ] [ -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 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.

Writability and Exclusive File Locking

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.

In a workspace where exclusive file locking is in effect (serial-development mode):

- The files that you are not actively working on are read-only. To work on a file, first enter a **co** or **anchor** command; this makes the file writable and places an exclusive file lock on it.

  (You *can* **keep** a file before making it writable and modifying it; but that probably doesn't make much sense.)

- If any other workspace based on the same backing stream already has a file under active development (**anchor**, **co**, **revert**, **keep**, **move**, **defunct**, **undefunct**), the resulting exclusive file lock prevents you from **keep**ing that file.

In a workspace where exclusive file locking is not in effect (parallel-development mode), all files are writable and existing locks don't affect your ability to **keep** files. See *Serial Development through Exclusive File Locking* on page 25 of the *AccuRev Concepts Manual*.

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 AC_EDITOR_GUI (or named in environment variable EDITOR, or in a system-dependent default editor).

–d Select all elements in your default group.

–E <*element-type*>
Change the element type to **text** 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. See *Controlling the Element Type* in the description of **add**.

–R Recurse into the directory specified in <*element-list*>, and keep all 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. There is no provision for comment lines in a list-file.

If you use this option, you cannot also specify an <*element-list*>.

<element-list>
One or more element names, separated by whitespace. If you specify a list of elements in this way, you cannot also use the **–l** option.

–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

Common Command Options on page 22, **add**, **mktrig**

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 make various changes to the specified dynamic stream:

- With no **–k** option, the lock:
  - 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**, users are prevented from promoting versions *from* the specified stream to other streams.
- With **–kt**, users are prevented from promoting versions *to* the specified stream from other streams.

Usage of both **–kf** and **–kt** in the same command is valid.

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. 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.

A lock on a stream does not prevent the stream from being modified with **chstream** or hidden 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.

Removing Locks

Use the **unlock** command to remove an existing lock. Placing a new lock on a stream automatically removes any existing lock on the stream.

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: enter a comment interactively, using the text editor named in environment variable AC_EDITOR_GUI (or named in environment variable EDITOR, or in a system-dependent default editor).

–kf
: ("from" lock) Disallow promotions *from* the stream. If you omit this option, promotions are disallowed *to* the stream.

–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.

–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.

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 -e mary tulip_dvt
```

See Also

unlock, **promote**, **chstream**, **remove**

lsacl show access control list information

Usage

```
accurev lsacl [ -fx ] { depot | stream }
```

Description

The **lsacl** command lists the access control list (ACL) entries that pertain to depots, or the entries that pertain to streams.

Options

–fx Produce command output in XML format.

Examples

List the ACLs that pertain to depots:

```
accurev lsacl depot
```

See Also

setacl, **chpasswd**, **mkuser**

lsrules — show the include/exclude rules for a workspace or stream

Usage

```
accurev lsrules [ -s <stream> ] [ -d ] [ -fx ]
```

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.

To restrict the listing to the rules that were explicitly set in your workspace (or stream), use the **–d** option.

The **lsrules** listing always includes the base rule (**incl \.**) set for the depot's base stream.

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.

–fx
: Produce command output in XML format.

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

incl, excl, show wspaces, show streams

merge — combine changes from another stream into the current version of an element

Usage

```
accurev merge [ -v <ver-spec> ] [ -G ] [ -K ] <element>

accurev merge –o [ -G ] [ -K ]

accurev merge –i [ -B ] [ -v <ver-spec> ] [ -s <stream> ] <element>

accurev merge –io [ -B ] [ -v <ver-spec> ] [ -s <stream> ]
```

Description

The **merge** command modifies your workspace's copy of an element in one or more of the following ways:

- **Content merge**: combining the contents of your workspace's copy with the contents of the version currently in the backing stream.
- **Element name merge**: resolving a discrepancy between the simple name ("leaf name") of the element in your workspace and its name in the backing stream.
- **Path merge**: resolving the discrepancy that the element's parent directory differs in the workspace and the backing stream. Note that this case occurs when the element is moved to a different parent directory. It does *not* occur when the parent directory is simply renamed (That's a change to the parent directory element, which has the "side effect" of changing the pathnames of the elements under it.)

(The term namespace-related applies to the latter two kinds of merge.)

You need to perform a merge when the element has **(overlap)** status in your workspace, which prevents it from being promoted to the backing stream. This occurs when the version in the backing stream is not a direct ancestor of the version in your workspace. (Previous merges count in this determination of an element's ancestry.)

merge ends by creating a new version in your workspace, incorporating the combined changes. You can promote the merged version to the backing stream.

Specifying Different "From" and "To" Versions

By default, the merge's from version is the version in the backing stream, and the to version is the version in your workspace. **merge** completes its work by creating a successor to the "to" version in your workspace.

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 an alternate "to" version. See the descriptions of these options for details.

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 which kind(s) or merge will be required — for example:

```
Path merge will be required.
Element name merge will be required.
```

merge then proceeds to perform the content merge, if necessary. It 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)
```

If there are no conflicting changes, **merge** announces:

```
Automatic merge of contents successful. No merge conflicts in contents.
```

If there are one or more coflicting changes, **merge** announces:

```
** merge conflicts in contents, edit needed **
```

After the content merge (if any), the path merge and element name merge are performed, if necessary. Each of these steps involves choosing one of three names. For example:

```
Path conflict for \.\dir02\sub03\file01.mary

Resolve path conflict by choosing path from:

(1) common ancestor: \.\dir02\sub02\ [eid=75]
(2) backing stream : \.\dir02\sub01\ [eid=68]
(3) your workspace : \.\dir02\sub03\ [eid=82]

Actions: (1-3) (s)kip (a)bort (h)elp

action ? [3] 2

Resolve name conflict by choosing name from:

(1) common ancestor: \.\dir02\sub01\file01.txt
(2) backing stream : \.\dir02\sub01\file01.john
(3) your workspace : \.\dir02\sub01\file01.mary

Actions: (1-3) (s)kip (a)bort (h)elp

action ? [3] 3
```

After merging the content and namespace-related changes, **merge** asks what you want to do next:

```
Actions: keep, edit, merge, over, diff, diffb, skip, abort, help

action ?
```

All these choices are related to the content merge

keep (default if there are no conflicting changes) 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 (default if there are some conflicting changes) Open a text editor on the merged file. Edit the conflict sections to resolve the conflicts and remove the separator lines.

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.

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 Checkouts or 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 **co –v** commands to do so. For example, if your backing

stream is named **gizmo_dvt** and you want to update elements **foo.c**, **bar.c**, and **read.me**, this set of commands accomplishes the "partial update":

```
accurev co -v gizmo_dvt foo.c
accurev co -v gizmo_dvt bar.c
accurev co -v gizmo_dvt read.me
```

(You'll probably want to use a script or a loop-processing command to make this task user-friendly.) If a file is active in your workspace, you can use the **merge** command instead of the **co** command to incorporate changes from the backing stream.

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 the graphical Merge tool to perform the merge if there are conflicts (which require human interaction).

–K If an overlap with the backing-stream version does not involve any conflicts (and thus, no 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).

–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.

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*>.

Preferences

The following preferences are implemented as environment variables.

AC_MERGE_CLI

A command string that **merge** will use to invoke an alternative merge program. (The default merge program is **acdiff3** in the AccuRev **bin** directory.) The command string must specify all four files involved in the merge, using these substitution patterns:

Pattern	Meaning in Command String
`%a`	common ancestor file
`%1`	version in backing stream
`%2`	file in workspace
`%o`	merge results (output) file

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**.

AC_MERGE_GUI (no longer supported!)

Prior to Version 3.0.2, AccuRev supported a preference AC_MERGE_GUI, which specified a graphical-file-comparison program to be invoked by the **Merge** GUI command. As of Version 3.0.2, you must specify such a program in the GUI window, with the command **File > Preferences > Merge**. See *Enabling the Diff and Merge Tools* on page 95 of the *AccuRev User's Guide (GUI)*.

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 98)
```

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;
  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**, *Common Command Options* on page 22

mergelist — determine which versions need to be promoted between streams

Usage

```
accurev mergelist -s <from-stream> -S <to-stream> [ -o ] [ -fl ]
   [ -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_jjp -S gizmo_3.4
```

This command lists all the versions that are active in the workspace stream **gizmo_3.4_jjp** — 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.

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.

–fl Display locations only (no status indicators or version-IDs).

–o Display only from-stream versions that overlap the corresponding to-stream versions (and thus require a merge before promotion).

–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. There is no provision for comment lines in a list-file.

If you use this option, you cannot also specify an <*element-list*>.

<element-list>

One or more element names, separated by whitespace. If you specify a list of elements in this way, you cannot also use the **–l** option.

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**, **promote**, **patchlist**

mkdepot — create a new depot

Usage

```
accurev mkdepot -p <depot-name> [ -l <storage-location> ] [ -Ci | -Cs ]
   [ -ke ] [ -w 96 | -w 128 } ]
```

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 **mkstream**).

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 1 of the *AccuRev Administrator's Manual*.

The slice top-level directory, which contains the depot's metadata, must be located on storage that is local to the machine where the AccuRev Server is running. However, after you have created a depot, you can relocate some or all of the depot's file storage area (the subtree under subdirectory **data**) to non-local storage. For more information, see *Storage Layout* on page 5 of the *AccuRev Administrator's Manual*.

Case-Sensitivity of Depots

A depot can be either case-sensitive (**–Cs** option) or case-insensitive (default, or **–Ci** option). 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 convert an existing case-sensitive depot to case-insensitive.

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 allows users to create two files (or subdirectories) in the same directory, with names that differ only in their case (e.g. **makefile** and **Makefile**).
- A case-insensitive depot does not allow two objects in the same directory to have names that differ only in their case.

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 54 of the *AccuRev Administrator's Guide*.

Options

–p <*depot-name*>
Specify a name for the new depot. The name must begin with a non-digit other than dot (**.**), and must not include either a slash (/) or a backslash (\).

–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). 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 cannot be changed with a subsequent **chdepot** command.

–l <*storage-location*>
Specify the full pathname of the directory where the depot is to be created. An error occurs if a file or a non-empty directory already exists at this pathname. Default: create the depot at a default location, established at AccuRev installation time.

–w 96 *or* **–w 128**
Create a depot in which each component of an element's pathname can be up to 95 characters (or 127 characters) long. By default, the maximum length of a pathname component is 63 characters. There is no maximum on the number of pathname components (i.e. on the depth of a depot's directory tree).

Note: a depot's **–w** setting cannot be changed with a subsequent **chdepot** command.

Examples

Create depot **brass** in the default location, enabling long pathname components:

```
accurev mkdepot -p brass -w 96
```

See Also

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. Groups implement security and establish development roles. For instance, you can allow/deny certain operations based on membership or non-membership in a group and you can also set up roles such as “developer” or “writer”.

Examples

Create an AccuRev group named **eng**:

```
accurev mkgroup eng
```

See Also

show groups, **show members**

mklinks — populate a workspace with links (Unix-specific)

Usage

```
accurev mklinks [ -R ] [ -O ] { <element-list> | <dir> }
```

Description

The **mklinks** command places links to elements into a link-based workspace, but does not add them to the default group. You should only need to use it if you have accidentally removed a link and wish to restore it.

Use the **update** command to keep workspaces up-to-date.

Options

–R Recursive.

–O Override the warning message that would otherwise appear, and overwrite existing files.

Examples

Restore the link to file **foo.c**:

```
accurev mklinks foo.c
```

See Also

update

mkref — create a new reference tree

Usage

```
accurev mkref -r <reftree-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, etc.; you use a workspace for creation of new versions (source-code development).

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 command you can use to modify the files in a reference tree is **update –r**. (And this is the *only* legitimate way to modify the files.) 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 61 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.

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**

- 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 <*reftree-name*>

Specify the name of the reference tree to be created. The name must begin with a non-digit other than dot (**.**), and must not include either a slash (/) or a backslash (\).

Unlike workspace names, reference tree names *can* be reused: you can specify a name that belonged to another reference tree, if it was subsequently renamed (**chref**). But you cannot specify the name of a reference tree that has been hidden with the **remove** command.

You cannot specify a name that is, or was, assigned to some workspace.

–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 line terminator, NL), or **w** (use the Windows line terminator, CR-NL).

Examples

Create a reference tree named **gizmo_snapl**, 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**

mksnap — create a new static stream

Usage

```
accurev mksnap -s <snapshot> -b <existing-stream> -t <time-spec>
```

Description

The **mksnap** command creates a new static stream (more commonly called a 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 **promote**s. But the contents of a snapshot never changes.

You can implement a "changeable snapshot" facility using a dynamic stream. See *Using the –t Option* in the description of **mkstream**.

Options

–s <*snapshot*>
: (required) Specify a name for the new snapshot. The name must begin with a non-digit other than dot (**.**), and must not include either a slash (/) or a backslash (\).

–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: e.g. **2001/08/07 20:27:15**
  - Time keyword: **now**
  - Transaction number as a positive integer: e.g. **146** or **23965**

  You cannot specify a time that is before the creation of the existing stream, or after the current time.

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 dynamic stream

Usage

```
accurev mkstream -s <stream> -b <backing-stream> [ -kp ] [ -t <time-spec> ]
```

Description

The **mkstream** command creates a new dynamic stream, based on an existing stream (termed the backing stream). A stream and its backing stream are sometimes described as "child" and "parent" streams.

Unlike a snapshot (static stream), a dynamic stream changes over time. Elements can be added to the stream, deleted, or moved to different pathnames. New versions of the elements can be created in the stream.

A newly created dynamic stream contains the same set of element versions as its backing stream. (With **–t**, it's 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 Dynamic Stream

The **promote** command places a new version of one or more elements in a dynamic stream. This is termed an "active change", because it's the result of an explicit command (**promote**), and is recorded in the depot's 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 **promote**d out of the stream, or are **purge**d from the stream.

> Note: 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.

A stream's active elements do not experience any of the "passive" changes described in the next section.

Passive Changes to a Dynamic Stream

In a dynamic stream, elements that are not currently active can still change, "passively". A dynamic stream inherits changes automatically from its backing stream. If a new version of **foo.c** is promoted to a backing ("parent") stream, this new version automatically appears in all of the "child" dynamic 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.

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 must begin with a non-digit other than dot (**.**), and must not include either a slash (/) or a backslash (\).

  You cannot specify a name that was assigned to another stream, even if that stream was renamed (**chstream**) or hidden (**remove**). This restriction applies even if the other stream was associated with a different depot. Once a name has been assigned to a stream, it remains permanently associated with that stream.

–b <*backing-stream*>
: Specify an existing stream, on which the new stream will be based.

–kp
: Make the new stream a pass-through stream. See *Pass-Through Streams* on page 18 of the *AccuRev Concepts Manual*.

–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: e.g. **2001/08/07 20:27:15**

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

You cannot specify a time that is before the creation of the backing stream, or after the current time.

See *Using the –t Option* above.

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
```

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
```

See Also

chstream, **lock**, **mkstream**, **promote**, **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

(See also: *AccuRev Triggers* on page 44 of the *AccuRev Adminisrator's Guide*.) 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.

Note: in the discussion below, we'll often use "trigger script" or "script" instead of the long-winded "trigger program or trigger script".

mktrig does not make a copy of the trigger script. It simply records the location at which the AccuRev will look for the script when 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 mount or Windows network share). If triggers will fire on both Windows and Unix machines, be sure to read *Notes on Triggers in Multiple-Platform Environments* on page 47 of the *AccuRev User's Guide (GUI)*.

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) or **System** (Windows).
- The script executes on the server machine, with the server-account identity.
- For certain operations (e.g. updating a reference tree), the server account must also have an AccuRev user identity. Be sure that the environment created for the server account includes the environment variable ACCUREV_PRINCIPAL, set to some name in AccuRev's user registry.

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 Built-in 'client_dispatch_promote' Trigger Procedure

For a **pre-promote-trig** trigger, you can invoke a built-in trigger procedure that implements the transaction-level integration between AccuRev and Dispatch. The integration procedure records the transaction number of each **promote** command in the **affectedFiles** field of a particular issue record.

Instead of naming a program or script file in the **mktrig** command, you use a special keyword:

```
accurev mktrig pre-promote-trig client_dispatch_promote
```

As with any **pre-promote-trig** trigger, this applies to a particular depot. You can specify the depot with the **–p** option.

For more on this integration — and on the change-package-level integration introduced in Version 3.5 — see *Integrations Between Configuration Management and Issue Management* on page 45 of the *Dispatch Issue Management Manual*.

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 apply to *all* depots, and 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** scrips 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.

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 44 of the *AccuRev Administrator's Manual*.

Dispatch Triggers

AccuRev's issue management facility, Dispatch, also supports triggers. See *The server_dispatch_post Trigger* on page 4 of the *Dispatch Issue Management Manual*.

Options

–p <*depot-name*>
Specify the depot in which the trigger is to be enabled.

Examples

Execute Perl script **elem-type.pl** before each **add** transaction:

```
accurev mktrig pre-create-trig elem-type.pl
```

Execute Perl script **addheader.pl** before each **keep** transaction:

```
accurev mktrig pre-keep-trig addheader.pl
```

Enable the transaction-level integration between AccuRev and Dispatch in depot **talon**:

```
accurev mktrig -p talon pre-promote-trig client_dispatch_promote
```

See Also

add, **keep**, **rmtrig**, **show triggers**

AccuRev Triggers on page 44 of the *AccuRev User's Guide (GUI)*

mkuser — register a new username

Usage

```
accurev mkuser <principal-name> { -kf | -kd } [ <password> ]
```

Description

The **mkuser** command defines a new user by creating a new principal-name in AccuRev's user registry. The user is licensed in either of these ways:

- can use both AccuRev configuration management and issue management (Dispatch) ("full" license)
- can use issue management (Dispatch) 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.

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.

Principal-names and Usernames

AccuRev determines your principal-name by examining the environment variable ACCUREV_PRINCIPAL. If this variable is not set, it uses your operating system username:

- Under Unix, it uses the value of the environment variable USERNAME or USER.
- Under Windows, it uses the value of the environment variable USER or the your username entry in the Windows registry.

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 password is considered to be an "authorized user" by the AccuRev ACL facility.

Don't assign a password with the **mkuser** command unless you are creating a principal-name for yourself. For other users, create the principal-name with **mkuser**, but have the user assign himself a password with **chpasswd**. This ensures that the user's **authn** file is created in the correct location. For more information, see *Password Storage and Password-Change Procedure* on page 44.

Options

–kf (default) Create a user licensed for both for both configuration management and issue management (Dispatch).

–kd Create a user licensed for issue management (Dispatch) only.

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 Dispatch user named **jane_doe**, with password **cv78w**.

```
accurev mkuser -kd jane_doe cv78w
```

See Also

chpasswd, **setlocalpasswd**, **chuser**, **show users**, **setacl**, **lsacl**

mkws — create a new user workspace

Usage

```
accurev mkws -w <workspace-name> -b <backing-stream> -l <location>
   [ -k <kind> ] [ -e <eol-type> ] [ -i ]

accurev mkws -w <workspace-name> -kl -r <linking-reftree>
   -l <location>
```

Description

The **mkws** command creates a workspace, in which you can work with the version-controlled files (elements) in a particular AccuRev depot. 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 depot 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).

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 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 filesystem, whether local or remote. Example: you create a workspace at location **/opt/workspaces/derek**; an error occurs if a remote filesystem 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*.

- Version-controlled text files in the workspace tree use the line terminator appropriate for your machine's operating system: NL for Unix, 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 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 **remove wspace** 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. **remove** merely hides the workspace, and attempts to create another workspace with the same name will fail.

Link-based Workspaces

On Unix systems, you can create a link-based workspace, using the **–kl** option. This requires the use of an existing reference tree, specified with **–r**, to link against. Reference trees are created with the **mkref** command.

Converting an Existing Directory Tree into a Workspace

The location you specify with the **–l** 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 both 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 must begin with a non-digit other than dot (**.**), and must not include either a slash (/) or a backslash (\). **chws** adds the suffix _<*principal-name*> to the name you specify (if you don't include the suffix yourself).

You cannot specify a name that was assigned to another workspace, even if that workspace was renamed (**chws**) or hidden (**remove**). This restriction applies even if the other workspace was associated with a different depot. Once a name has been assigned to a workspace, it remains permanently associated with that workspace.

You cannot specify a name that is, or was, assigned to some reference tree.

–r <*linking-reftree*>

(Unix-only) When creating a link-based workspace with **–kl**, you must use this option to specify the existing reference tree whose files will be the targets of the links.

–k <*kind*>

Specify the kind of workspace to be created (–kd and –kl are mutually exclusive; –ka and –ke are mutually exclusive):

–kd: Create a workspace with the default storage scheme: fully populated with copies of versions.
–kl: (Unix only) Create a workspace consisting of links to a reference tree. See *Link-based Workspaces* on page 114.

–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.

As of Version 3.5, you cannot create a sparse workspace. See the descriptions of **incl** and **excl**.

–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 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

mkref, **show streams**, **start**, **incl**, **excl**

move — move or rename elements

Usage

```
accurev move <file-element-name> <new-name>

accurev move <file-element-name> <destination-directory>

accurev move <file-element-name> <destination-directory>/<new-name>

accurev move <directory-element-name> <destination-directory>

accurev move -e <eid> <depot-relative-pathname>
```

Description

Note: you can spell this command name either **move** or **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. Specifying a directory makes it a subdirectory of the <*destination-directory*> you specify.

You can use the **–e** option to specify an element that does not currently have a pathname in your workspace — for example, an element stranded by the **defunct** command. You may need to use a command such as **hist** or **stat –fe** to determine the element-ID of such an element.

The **move** command creates a new version in the workspace stream, and makes the element active in the workspace stream. If a file's status is **(modified)** before the **move**, it remains **(modified)** after the **move**; use the **keep** command to create a new version that records the file's content changes.

Private Workspaces, Pathname Handling, and 'Purging' Changes

The **move** command fits neatly into AccuRev's development architecture, wherein all changes originate in a private, isolated workspace. Thus, **move** is like **keep**: the pathname change affects only your own workspace. People working in other workspaces continue to see the element at its old pathname. As with **keep**, you use **promote** to propagate the change to your workspace's backing stream.

As of Version 3.8, using **move** does not "reserve" the element's old name. After renaming file **blue.java** to **colors.java**, for example, you are free to create a new element named **blue.java**. The new element is unrelated to the original element — the two elements merely have had the same pathname, at different times.

The AccuRev development architecture also allows you to discard the changes you've made to an element, instead of promoting them. The **purge** command performs this "undo" operation, returning an element to its state when you last updated the workspace (or when you last promoted the element). If you use **move** to rename an element — one time, or multiple times — then invoke

the **purge** command on the renamed element, the element reverts to its name before the first **move**.

You can also combine the scenarios of the two preceding paragraphs: (1) rename **blue.java** to **colors.java**, (2) create a second element named **blue.java**, and (3) purge the changes to the element now named **colors.java**. In this case:

- The second element remains visible in your workspace, as **blue.java**. It also remains active in the workspace (that is, remains in the default group). The **purge** command does not affect this second element at all.
- The purged element, most recently named **colors.java**, disappears from your workspace entirely. It does not revert to its original name, **blue.java**, because that name is now being used by the second element. As usual, the **purge** operation causes the element to stop being active in the workspace (that is, removes the element from the default group).

Although the first element no longer appears in your workspace, you still may be able to perform various operations on it, using the **–e** option to certain commands, such as **cat**, **hist**, and **move**.

Exclusive File Locking

The **move** command fails if both of these conditions hold:

- You are in a workspace where exclusive file locking is in effect (serial-development mode).
- Some other workspace based on the same backing stream already has any of the specified files under active development (**anchor**, **co**, **revert**, **keep**, **move**, **defunct**, **undefunct**).

See *Serial Development through Exclusive File Locking* on page 25 of the *AccuRev Concepts Manual*.

Options

–e <*eid*>

Operate on the element with the specified element-ID. Use a depot-relative pathname to specify the new pathname of the element.

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**:

```
accurev move foo.c subdir/bar.c
```

See Also

promote

name — list the name of the element with the specified element-ID

Usage

```
accurev name [ -v <ver-spec> ] -e <eid>
```

Description

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.

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 26 for a description of the forms that <*ver-spec*> can take.

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 ] -v <ver-spec> <element>
```

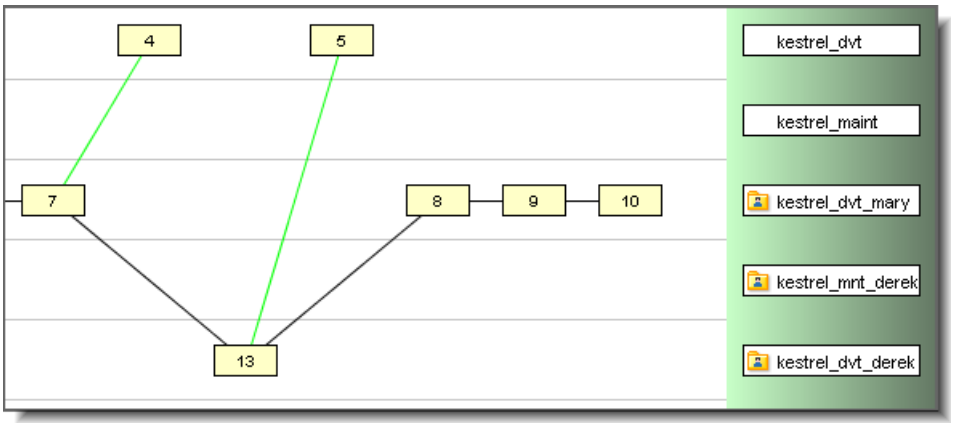
Description

The **patch** command is a variant of **merge**:

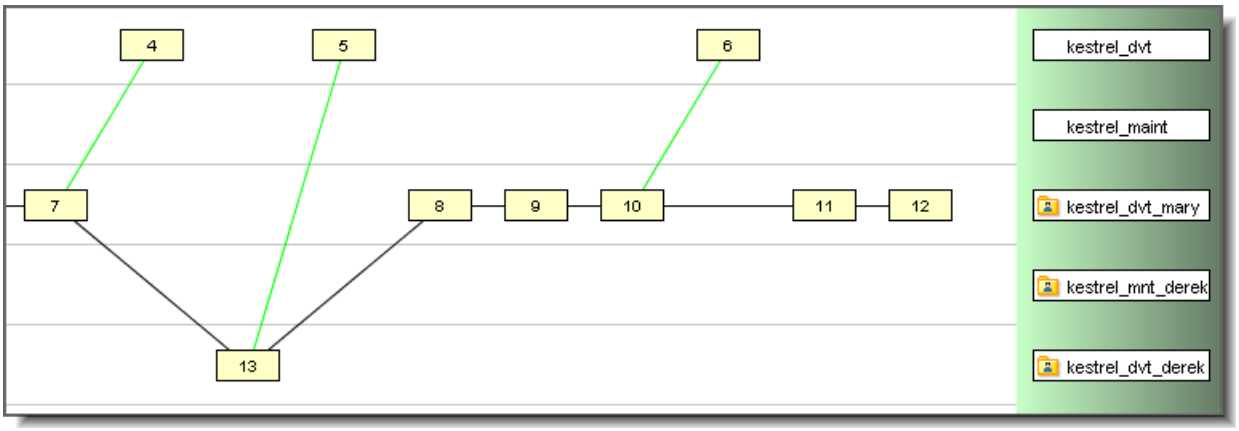
- **merge** incorporates the entire set of changes between the from version and the common ancestor version.
- **patch** incorporates just the set of "recent changes" in the from version. It determines this set of changes by starting at the from version and scanning backward through the file's ancestry. 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.

Examples:

- 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 **dvt_derek/13**. Then she proceeded to create versions **dvt_mary/8** through **dvt_mary/10**. When you patch version **dvt_mary/10** into your workspace, AccuRev searches backward through the element's ancestry, and includes the changes in all the versions recently created in the same workspace: **dvt_mary/8**, **dvt_mary/9,** and **dvt_mary/10**. It doesn't include the predecessor of **dvt_mary/8** — version **dvt_derek/13** — because it was created in a different workspace.



- Suppose Mary started working as described above, but proceeded a bit differently: she created versions **dvt_mary/8** through **dvt_mary/10**, promoted her work to the backing stream, then created versions **dvt_mary/11** and **dvt_mary/12**. In this case, when you patch version **dvt_mary/12** into your workspace, AccuRev decides that only the versions since the promotion — versions 11 and 12 — contain "recent changes". The idea is that a promotion typically marks the end of a programming task, not an intermediate checkpoint.



The basis version is indicated in the **keep** transaction that records a **patch** command in the repository:

```
> accurev patch -v 6/7 foo.c
transaction 106; keep; 2004/10/13 13:38:39 ; user: jjp
  # patched
  version 7/3 (7/3)
  ancestor: (7/2) patched from: (5/2)-(6/7)
```

In this command, version 6/7 was patched into version 7/2, and the result was kept as version 7/3. The backward search through the file's ancestry identified 5/2 as the basis version. So the patch encompasses the series of versions in workspace #6 that follow version 5/2, up to and including version 6/7.

See the description of **merge** for more information.

AccuRev keeps track of patches separately from merges.

Options

–G Use the graphical Merge tool to perform the operation if there are any conflicts (which require human interaction).

–v ***<ver-spec>***
Specify the version whose changes are to be incorporated into your workspace's copy of the element.

Examples

Patch in the change that was made in version 4/3 of **foo.c**:

```
accurev patch -v 4/3 foo.c
```

See Also

diff, **merge**, **mergelist**, **patchlist**

patchlist — list versions that need to be patched into the workspace's version

Usage

```
accurev patchlist -v <from-ver-spec> [ -V <to-ver-spec> ]
   [ <element-list> ]
```

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 an element list, all the elements in the "to" workspace/stream are considered.

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. 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.

You can specify the "from" and "to" versions by version-ID (**gizmo_dvt/5**) or by stream (**gizmo_dvt**).

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.

–V ***<to-ver-spec>***
: Specify the version that you wish the changes to be incorporated into.

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

patch, **merge**, **mergelist**

ping — test client-server communications link

Usage

```
accurev ping

accurev ping 1 [ <block-size> ]

accurev ping 2 [ <block-size> <cycles> <transfers-per-cycle>
   <pause-between-cycles> <report-interval> ]
```

Description

The **ping** command tests the communications link between your client machine and the AccuRev Server machine, by sending data blocks in both directions and reporting performance statistics. You can perform two different tests:

- In a speed test (type 1), **ping** performs a series of data transfers — from server to client, then from client to server. The transfers are increasingly large sets of data blocks; **ping** reports the overall throughput for each set of blocks:

```
> accurev ping 1 5
Test ID: 1
Blocksize: 32
----- receive speed test -----
ping:            1 blocks           800 bytes/sec
ping:            2 blocks          1279 bytes/sec
ping:            4 blocks          3200 bytes/sec
ping:            8 blocks          5120 bytes/sec
ping:           16 blocks          4654 bytes/sec
...
ping:        32768 blocks         52536 bytes/sec
----- send speed test -----
ping:            1 blocks           680 bytes/sec
ping:            2 blocks          2064 bytes/sec
ping:            4 blocks          2723 bytes/sec
ping:            8 blocks          4129 bytes/sec
ping:           16 blocks          8126 bytes/sec
...
ping:         2048 blocks          5236 bytes/sec
----- tests completed -----
```

  You can specify the data-block size using a command-line argument.

- In a continuity test (type 2), **ping** alternates between sending and receiving sets of data blocks. Each iteration is called a "cycle": sending a set of blocks to the AccuRev Server, then receiving a set of blocks from the Server, then pausing. Periodically, **ping** reports on the overall throughput:

```
> accurev ping 2 8 25 50 1 5
Test ID: 2
```

```
Blocksize: 256
Loop 1: 25
Loop 2: 50
Wait Time: 1
Report Frequency: 5
----- continuity test -----
    5.88 sec        4 cycles      200 xfers      0.7 cycles/sec     34.0 xfers/sec         8710 bytes/sec
    6.94 sec        4 cycles      200 xfers      0.6 cycles/sec     28.8 xfers/sec         7378 bytes/sec
    6.29 sec        4 cycles      200 xfers      0.6 cycles/sec     31.8 xfers/sec         8141 bytes/sec
    4.66 sec        3 cycles      150 xfers      0.6 cycles/sec     32.2 xfers/sec         8246 bytes/sec
    5.90 sec        3 cycles      150 xfers      0.5 cycles/sec     25.4 xfers/sec         6511 bytes/sec
    4.70 sec        3 cycles      150 xfers      0.6 cycles/sec     31.9 xfers/sec         8175 bytes/sec
    5.85 sec        3 cycles      150 xfers      0.5 cycles/sec     25.6 xfers/sec         6565 bytes/sec
    1.00 sec        1 cycles       50 xfers      1.0 cycles/sec     50.0 xfers/sec        12787 bytes/sec
    1.00 sec        1 cycles       50 xfers      1.0 cycles/sec     50.0 xfers/sec        12787 bytes/sec
----- tests completed -----
```

You can use command-line arguments to specify the data-block size and the cycle parameters.

Options

All of the **ping** command-line arguments are optional. Specifying no arguments at all (**accurev ping**) is equivalent to the command **accurev ping 1 11**: a speed test (test-ID = 1) is performed, using a block size of 2^11 = 2048 bytes.

If specified, the command-line arguments must appear in this order:

<test-ID>

(default: 1) Either **1** (speed test) or **2** (continuity test).

<block-size>

(default: 11, specifying a block size of 2^11 = 2048 bytes) A power of 2, specifying the size of the data blocks to be transferred.

The following arguments apply only to continuity tests (test-ID = 2).

<cycles>

The number of receive-send cycles to be performed.

<transfers-per-cycle>

The number of transfers to be included in each cycle. For example, specifying **20** makes a cycle consist of receiving 20 data blocks from the Server, followed by sending 20 data blocks to the Server.

<pause-between-cycles>

The number of seconds to pause between cycles.

<report-interval>

The reporting interval, in seconds. At the end of each reporting interval, **ping** reports on the cycles completed during that interval.

Examples

Perform a speed test, using a block size of 128 (2^7) bytes:

```
accurev ping 1 7
```

Perform a continuity test with 15 cycles, each of which consists of receiving 75 blocks and then sending 75 blocks. Wait 2 seconds between each cycle, and report throughput statistics every 10 seconds. Each data block should be 512 (2^9) bytes.

```
accurev ping 2 9 15 75 2 10
```

pop — copy files into a workspace or reference tree

Usage

```
accurev pop [ -O ] [ -R ] [ -v <ver-spec> -L <location> ]
   { -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. For example, if you never use a sparse workspace and you never accidentally delete any files, then you may never need **pop**.

It's appropriate to use **pop** in these cases:

- **Using a sparse workspace**: When you need to work with a depot that contains many hundreds or thousands of elements, you can use a sparse workspace, in order to save time and disk space. Initially, the workspace is an empty directory — no data is automatically loaded into it from the depot. You can then use **pop** to load specific subdirectories, or even individual files.

  Note: as of Version 3.5, you cannot create a new sparse workspace or change an existing workspace to be sparse. The sparse workspace facility has been superseded by the include/exclude facility. See the **incl** and **excl** reference pages.

- **Accidental deletion of files**: Deletion happens. There's nothing AccuRev can do if you delete files that you've been editing, but haven't 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 haven't 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's the version that was loaded (or left alone) by that update.
- For a file that you are loading into a sparse workspace for the first time, it's the version that *would* have been loaded by your most recent update.
- For a file that you have modified and preserved with **keep**, but then accidentally deleted, it's the most recently kept version.

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 155.

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 copy operation. Use depot-relative pathnames to specify the elements to be copied. (See *Depot-Relative Pathnames* on page 154.) The current directory must not be within a workspace.

The best way to copy *all* the versions in a particular stream is to create a reference tree (see **mkref**). To select a specific version of a file for development use inside your workspace, use the **co** command.

Options

–O Override: suppress the warning message that would appear, and overwrite existing files.

–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.

–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. There is no provision for comment lines in a list-file.

If you use this option, you cannot also specify an <*element-list*>.

<element-list>

One or more element names, separated by whitespace. If you specify a list of elements in this way, you cannot also use the **–l** option.

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.

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
```

See Also

update

promote — propagate a version from one stream to another stream

Usage

```
accurev promote [ -c <comment> ] [ -d ] [ -k ] [ -p ] [ -O ] [ -K ]
   [ -I <issue-number> ] [ -g ]
   [ -l <list-file> | <element-list> | -t <trans-number> | -e <elem-ID> ]

accurev promote [ -c <comment> ] -s <from-stream> [ -S <to-stream> ]
   [ -I <issue-number> ] [ -g ]
   [ -l <list-file> | <element-list> | -t <trans-number> | -e <elem-ID> ]
```

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 a developer's workspace stream with the **keep** command, to the backing stream. This has the effect of taking the developer's private changes and making them public.

Other changes that can be promoted include: renaming and moving of elements, creation of new elements, and defuncting of elements.

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.

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, use the **–t** option to specify the transaction in which that older version was created. Note that the command 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
```

Promotion and the Default Group

Promoted elements are 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.

Exclusive File Locking

In a workspace where exclusive file locking is in effect, **promote** returns a file returns to being read-only and **keep**-disabled. See *Serial Development through Exclusive File Locking* on page 25 of the *AccuRev Concepts Manual*.

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 **keep**ing the merged file, you can promote it to the backing stream.

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 isn't its parent. Use both the **–s** and **–S** options, to specify the "from" and "to" streams.

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.

Promotion-Related Triggers

AccuRev defines two 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.

For more information on triggers, see the **mktrig** reference page and *AccuRev Triggers* on page 44 of the *AccuRev Administrator's Guide*.

Integrations with Dispatch

Invocation of the **promote** command can activate one or both of the integrations between AccuRev's configuration management and issue management facilities. Before the **promote** operation is performed, you are prompted to enter the number of a Dispatch issue record:

```
Validating elements.
Please enter issue number ?:
```

After the elements are promoted, the specified Dispatch issue record is updated in one or both of these ways:

- The transaction-level integration updates the **affectedFiles** field of the issue record.
- The change-package-level integration updates the change package (Changes page) of the issue record.

You can bypass this prompting for an issue number (for example, in a script) by using the **–I** option to specify an issue number. You may also need to use the **–g** option, as described below.

Updating an Existing Change Package Entry. If an element being promoted already has an entry in the specified change package, AccuRev attempts to combine the element's new change set entry with the existing change set entry. This attempt can fail with either of these messages:

- "Change package gap" — The head version of one change set entry is not the same as, but *is* an ancestor of, the base version of the other change set entry.
- "Change package merge required" — There is no direct ancestry relationship between the head versions of the two change set entries.

For details on the "change set arithmetic" terms used above, see *Updating of Change Package Entries* on page 43 of the *Dispatch Issue Management Manual*. Also, see *Integrations Between Configuration Management and Issue Management* on page 45 of the *Dispatch Issue Management Manual*.

Options

–d Select all elements in the default group of the workspace (or more generally, the child stream).

–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 AC_EDITOR_GUI (or named in environment variable EDITOR, or in a system-dependent default editor).

–e <*eid*>
Operate 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.

–g Automatically "span the gap" in a change package entry. See *Integrations with Dispatch* above.

–I <*issue-number*>
Bypasses the prompting for an issue record by the transaction-level integration (see *Integrations with Dispatch* above). The integration uses the number you specify after **–I**.

–k Select all kept elements. (Not valid with **–s**.)

–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.

–p Select all pending elements. (Not valid with **–s**.)

–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.

–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.

–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. There is no provision for comment lines in a list-file.

If you use this option, you cannot also specify an <*element-list*>.

<element-list>

One or more element names, separated by whitespace. If you specify a list of elements in this way, you cannot also use the **–l** option.

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>*

Promote all the versions created in the specified transaction. See *Promoting an 'Old' Version* above.

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
```

See Also

Common Command Options on page 22, **co**, **keep**, **merge**, **mktrig**

purge — undo all of a workspace's changes to an element

Usage

```
purge [ -s <stream> ] { -l <list-file> | <element-list> | -e <eid> }
```

Description

The **purge** command undoes all of the changes you have made to an element since you activated it in your workspace — i.e. 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 — i.e. 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 **purge**d, 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 155.

Effect of "purge" on "defunct" 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 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!

Options

–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.

–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. There is no provision for comment lines in a list-file.

If you use this option, you cannot also specify an <*element-list*>.

<element-list>

One or more element names, separated by whitespace. If you specify a list of elements in this way, you cannot also use the **–l** option.

If you use **–s** to purge changes from a non-workspace stream, then each name in <*element-list*> must be a depot-relative pathname.

–e <*eid*>

Operate on the element with the specified element-ID. You can use this option instead of specifying a list of elements. This is useful for removing stranded elements from the default group. Use **stat –i** to list stranded elements.

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**, **undefunct**

reactivate — make a reference tree, stream, user, or workspace visible again

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 reference tree, stream, user, group, or workspace that was previously hidden (with the **remove** command) visible again.

You cannot use **remove** and **reactivate** on elements. The comparable commands for elements are **defunct** and **undefunct**.

Examples

Make stream **gizmo** visible again, after having previously hidden it:

```
accurev reactivate stream gizmo
```

See Also

defunct, **undefunct**, **remove**, **show**

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 25 of the *AccuRev Administrator's Guide*.

See Also

archive, **unarchive**

remove — hide a reference tree, stream, user, or group

Usage

```
accurev remove ref <reftree>
accurev remove stream <stream>
accurev remove user <principal-name>
accurev remove group <group-name>
[deprecated] accurev remove wspace <workspace-name>
```

Description

Note: the **remove wspace** command is now deprecated. Use the **rmws** command instead. In particular, you can use the **rmws –s** command to hide another user's workspace.

The **remove** command hides a reference tree, stream, user, group, or workspace. It does not actually remove the object from the depot — that would violate AccuRev's TimeSafe property. Instead, it renders the object invisible and — depending on the kind of object — restricts usage of the object:

- **ref** — A removed reference tree cannot be updated (**update**), but can be renamed (**chref**).
- **stream** — A removed stream can still be referenced in various commands, with the **–s** or **–v** option.
- **user** — Removing a user from the AccuRev user registry effectively frees an AccuRev license for use by another user — either create a new principal-name (**mkuser**) or reinstate a previously removed principal-name (**reactivate**). If AccuRev determines that your principal-name has been removed, you won't be able to run most commands.
- **group** — Removing a group simply hides it. There is no effect on the users in the group.
- **workspace** — see the *rmws* description

To see hidden objects, use the **–fi** or **–fI** option to the **show** command.

If you make a mistake creating an object, you may want to **remove** it. But since this command merely hides the object, without actually deleting it from the depot, you cannot create a new object with the same name.

Examples

Hide stream **gizmo**:

```
accurev remove stream gizmo
```

Hide one of your own workspaces:

```
accurev remove wspace gizmo_maint
```

Deactivate user **tom**:

```
accurev remove user tom
```

See Also

reactivate, **show**, **rmws**

replica — synchronize a replica repository

Usage

```
accurev replica sync
```

Description

The **replica** command brings your local replica repository up to date with 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 this CLI command to bring your local replica repository up to date:

```
accurev replica sync
```

This command transfers database transactions from the master repository to the replica repository. It 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.

See Also

Replication of the AccuRev Repository in the *AccuRev Administrator's Guide*

revert — "undo" a promote transaction

Usage

```
accurev revert -t <transaction-number>
```

Description

The **revert** command makes it easy to undo the effects of a **promote** transaction. You can't *literally* undo any transaction, since that would violate AccuRev's TimeSafe property. So **revert** does the next best thing:

- It determines which elements were involved in the transaction you specify with **–t**. This must be a **promote** transaction.
- For each of these elements, it checks out (**co –v** command) a version that *precedes* the version that was promoted. The version that gets checked out is the version that would have been reinstated if you had originally discarded your work instead of making it public — that is, if you had performed a **purge** instead of a **promote**.
- It displays the exact command ("promote –O –t...") you can use to promote the old versions, ignoring the overlap that normally would require a merge. Note that these versions are not promoted automatically.

Typically, you'll execute **revert** in the workspace where you executed the "bad" **promote**. In this case, the workspace is "rolled back" to its state before you modified, kept, and promoted those elements. (The rollback involves only those particular elements, not the entire workspace.)

Be careful — revert-then-promote effectively discards work. The older the transaction you revert, the more likely it is that the discarded work will include changes made in other workspaces, not just your own workspace.

> Note: as of AccuRev Version 3.0.1, you can only revert changes to the contents of files; you cannot revert changes to filenames or to the depot's directory structure.

Timestamps on Copies of Old Versions

The **revert** command copies old versions of one or more elements 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.) Setting this environment variable may defeat an optimization in the **stat** command. See *Optimized Search for Modified Files* on page 155.

Exclusive File Locking

In a workspace where exclusive file locking is in effect (serial-development mode):

- The files that you are not actively working on are read-only. Reverting to an old version of an element creates a writable file in your workspace and places an exclusive file lock on the element.
- If any other workspace based on the same backing stream already has an element under active development (**anchor**, **co**, **revert**, **keep**, **move**, **defunct**, **undefunct**), the resulting exclusive file lock prevents you from **revert**ing to an old version of that element.

In a workspace where exclusive file locking is not in effect (parallel-development mode), all files are writable and existing locks don't affect your ability to **anchor** files. See *Serial Development through Exclusive File Locking* on page 25 of the *AccuRev Concepts Manual*.

Options

–t <*transaction-number*>

(required) Select all the elements involved in the specified transaction. (You must specify the transaction by number, not by time.)

Examples

"Undo" transaction 489, restoring to the workspace the version that preceded each promoted version:

```
accurev revert -t 489
```

See Also

co, **anchor**

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**, **show members**

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 ]
```

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.

Description

The **rmtrig** command removes a trigger.

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 — hide a workspace

Usage

```
accurev rmws [ -s ] <workspace-name>
```

Description

The **rmws** command hides the specified workspace. More precisely, it hides the workspace stream in the depot, but does not make any change to the workspace tree on your disk. The hidden workspace won't appear in **show wspaces** listings, unless you use the **–fi** or **–fI** option.

A hidden workspace can still be used in commands that reference the workspace stream — for example:

```
accurev remove wspace talon_dvt_jjp
  ...
accurev stat -s talon_dvt_jjp -a                    (succeeds)
accurev promote -s talon_dvt_jjp -d                 (succeeds)
accurev cat -v talon_dvt_jjp/13 /./src/base.h       (succeeds)
```

But if you go to a directory in the workspace tree, AccuRev will not recognize you as being "in a workspace". And you won't be able to **keep** new versions or perform other operations that use the workspace tree:

```
cd /home/jjp/talon_dvt
accurev files                                       (fails)
```

Use the **reactivate** command to make a hidden workspace visible and fully usable again.

Options

–s Use this option when you hide 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

Hide your own workspace, named **talon_dvt_<*your-principal-name*>**:

```
accurev rmws talon_dvt
```

Hide another user's workspace, specifying its complete name:

```
accurev rmws -s talon_dvt_mary
```

See Also

remove, **show wspaces**, **reactivate**

setacl — set an access control list entry

Usage

```
accurev setacl depot <depot-name>
   { anyuser | authuser | <group-name> } { none | all | clear }

accurev setacl stream <stream>
   { anyuser | authuser | <group-name> } { none | all | clear }
```

Description

The **setacl** command changes the access control list (ACL) for a depot or stream. An ACL controls the rights of one or more users to access the data within the depot or stream. By default, AccuRev is wide open: all users can access all depots and all streams within the depots.

ACLs control a user's ability to perform the following operations:

```
cat, co, diff, merge, mkws, pop, update
```

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.

Setting ACLs

All forms of the **setacl** command follow the same pattern:

- Specify the data structure:
  - **depot** <*name-of-depot*> sets an ACL that controls access to all the data within a particular depot.
  - **stream** <*name-of-stream*> sets an ACL that controls access to all the data within a particular stream 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.
- Specify the set of users:
  - **anyuser** specifies all users who *do not* have a password.
  - **authuser** specifies all users who *do* have a password.
  - <*name-of-group*> specifies all users in a particular AccuRev group. Only one group may be associated with a particular depot or stream: setting an ACL for one group removes any existing ACL for another group.
- Specify the access level to be granted:
  - **all** grants access to all the data in the specified data structure to the specified users.
  - **none** prohibits access to all the data in the specified data structure for the specified users.

  A group can only be granted access (**all**), not denied access (**none**).

Multiple ACLs can apply to the same user. For example, a user with a password is subject to an **authuser** ACL; that user might belong to a group that is named in a group ACL. When determining the rights of a user, AccuRev uses the following precedence order to resolve conflicts among multiple applicable ACLs:

group ACL (highest)
authuser ACL
anyuser ACL (lowest)

Removing ACLs

Using the keyword **clear** removes an existing ACL for the specified data structure and set of users. Also, setting an ACL for one group removes any existing ACL for another group.

When you remove an ACL with **clear**, some or all of the users may still be covered by another ACL. For example, you might remove an **anyuser** ACL, but some of those non-password users might belong to a group for which there is another ACL.

Examples

Grant access to depot **gizmo** to authenticated users only:

```
accurev setacl depot gizmo anyuser none

accurev setacl depot gizmo authuser all
```

See Also

lsacl, **chpasswd**, **mkuser**

setlocalpasswd create an authn file on the local machine

Usage

```
setlocalpasswd <password>
```

Description

The **setlocalpasswd** command creates (or replaces) a 1-line file named **authn** in subdirectory **.accurev** of your home directory. The contents of this file is the password you specify as an argument.

On Windows systems, your home directory is determined by concatenating the values of the environment variables HOMEDRIVE and HOMEPATH. On some versions of Windows, you must set these environment variables yourself; they are not set automatically by the operating system.

AccuRev client programs verify that your user identity is registered as a principal-name in the repository. If the principal-name is registered with a password, the client program makes sure the contents of the **authn** file matches the registered password.

The **mkuser** and **chpasswd** commands automatically perform the work of **setlocalpasswd** on the local machine. If you use several AccuRev client machines, you need to use **setlocalpasswd** on each other machine whenever you change your password on one of those machines.

Examples

Change your password on two AccuRev client machines, **able** and **baker**:

on machine **able** ...

```
accurev chpasswd ou812
```

on machine **baker** ...

```
accurev setlocalpasswd ou812
```

See Also

chpasswd, **mkuser**

show — list all depots, streams, workspaces, or slices

Usage

```
show locks
show [ -fx ] depots
show [ -f<format> ] groups
show [ -fx ] [ -g <group-name> ] members
show [ -f<format> ] refs
show [ -f<format> ] slices
show [ -f<format> ] [ -p <depot-name> ]
   [ -s <stream> ] [ -t <time-spec> ] streams
show [ -f<format> ] [ -p <depot-name> ] triggers
show [ -f<format> ] [ -v ] users
show [ -f<format> ] [ -a ] wspaces
```

Description

The **show** command displays information about all objects of a particular type:

- **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.
- **depots**: For each AccuRev depot, displays (**Depot**) the depot name, (**Depot#**) the depot number, and (**Slice#**) the slice number.
- **streams**: For each stream in the repository — or a subset of the streams specified by the **–p**, **–s,** 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, and (**Basis Time**) the basis time, if any, for a dynamic stream.

  The listing includes workspace streams, passthrough streams, and snapshots.
- **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.
- **wspaces** (workspaces): For each workspace that belongs to you — or for all workspaces in the repository — displays (**Workspace**) 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 level** and **update level**: The target level indicates how up-to-date the workspace *should* be; the update level indicates how up-to-date the workspace actually is. Usually, these two levels are the same. See *Target Level, Update Level, and Update Time* on page 170.

    - **workspace type**: One of the following: 1 (standard workspace), 5 (link-based, Unix-only), 9 (exclusive-file locking), 17 (anchor-required).
    - **Text-file EOL type**: One of the following: 0 (platform-appropriate), 1 (Unix style: NL), 2 (Windows style: CR-LF)
- **groups**: For each group, displays (**Group#**) the group number and (**Group**) the group name.
- **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.
- **users**: For each user, displays (**User**#) the unique user-ID number and (**User**) the principal-name.
- **locks**: Displays a line for each stream lock, in this form:

  `{to|from|all} <stream-name> [ {except|only} for <user-or-group-name> ]`
- **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.

Options

–a Display all workspaces. By default, **show** displays only workspaces that belong to you (i.e. to your principal-name).

–f <*format*>
–fi: Include hidden (removed) items in the listing.
–fI: Include hidden (removed) items, and include old definitions of all items.
–fx: Display command output in XML format.
–fv: Add a Kind column to **show users** output, indicating whether the user is licensed for use of "full" (both configuration management and issue management) or "dispatch" (issue management only).

–g <*group-name*>
Restrict a member listing to the specified group.

–p <*depot-name*>
Restrict a stream or trigger listing to the specified depot.

–s <*stream*>
Restrict the listing to the specified stream. (You must also use the **streams** argument.)

–t <*time-spec*>
Show the streams that existed at the specified time. You must also use the **–p** option to specify a depot. A time-spec can be any of the following:

- Time in <YYYY/MM/DD HH:MM:SS> format: e.g. **2004/06/07 20:27:15**
- Time keyword: **now**

- Transaction number as a positive integer: e.g. **146** or **23965**

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 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
```

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: 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 as 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
```

Run a program named **overboard** in the top-level directory of workspace **tulip_dvt**:

```
accurev start -w tulip_dvt -c overboard
```

See Also

mkws, **show wspaces**

stat — show the status of elements

Usage

Status of elements in your workspace:

```
accurev stat [ -f<format> ] [ -adkmnop ] [ -O ] [ -b ] [ -B ] [ -R ] [ -M ]
   [ -i ] [ -l <list-file> | <element-list> ]
```

Status of external files in your workspace:

```
accurev stat -x [ -f<format> ] [ -R ] [ -l <list-file> | <element-list> ]
```

Status of versions in a specified stream:

```
accurev stat -s <stream> [ -f<format> ] [ -adoi ]
   [ -l <list-file> | <element-list> ]
```

Status of defunct versions:

```
accurev stat -D [ -s <stream> ] [ -f<format> ]
   [ -l <list-file> | <element-list> ]
```

–f option format letters:

```
[ a | r ] [ f | d ] [ l ]
```

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 6 of the *AccuRev Concepts Manual* for a detailed description of virtual versions and real versions.

status indicators

(Not included if you use the **–a** option) One or more of the keywords listed below. There are several categories.

Presence of the element in the stream:

- **(defunct)** — the element has been marked for removal from the stream with the **defunct** command.
- **(external)** — the file or directory has not been placed under version control.
- **(missing)** — the workspace "should" include a version of this element, but doesn't. This occurs when you delete version-controlled files from the workspace.

Changes to the element in the stream:

- **(modified)** — (workspace only) the element has been modified in the workspace since the most recent **update** or **keep**. See *Optimized Search for Modified Files* below.
- **(kept)** — (workspace stream only) the element has been kept and not subsequently modified or promoted to the backing stream.
- **(member)** — the element is in the stream's default group.

Relationship to the backing stream:

- **(backed)** — the element has not been changed since the most recent **update** of your workspace. (Or you've made changes to the element, and have already **promote**d the changes to the backing stream.)
- **(stale)** — the element has changed in the backing stream, but the change has not been incorporated into the workspace with an update.
- **(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.

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
```

(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) or `\.\` (Windows) to indicate a depot-relative pathname:

```
/./src/commands/base.h                    (Unix 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 aboslute pathname)
c:\gizmo_dvt_jsmith\src\commands\base.h               (Windows aboslute 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 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**, **–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 doesn't consider files whose timestamps precede the workspace's update time (the time that the workspace was most recently updated).

Note: this optimization is not used in a search for external files (**stat –x**).

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 **stat** to silently ignore relevant 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.

Using 'update' to Improve 'stat' Performance

The timestamp optimization described above produces the greatest performance boost when it enables **stat** to ignore a large number of files based on their timestamps. If **stat** seems sluggish, try executing an **update** command. If the workspace contains a significant number of files whose timestamps fall between the previous update and the current time, the optimization will enable a subsequent execution of **stat** to ignore these 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.

–B Display the status of all versions in the backing chain of streams above the workspace stream. This will reveal any deep overlaps for the element.

–d Select only elements in the default group.

–D Select only defunct elements in the specified stream. You must specify the stream; it can be a workspace stream or a dynamic stream.

–f... 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.

–fa = display locations as absolute pathnames
–fr = display locations as relative pathnames (relative to the current working directory)

–ff = display files only
–fd = display directories only
–fl = display locations only (no status indicators or version-IDs)

You can use appropriate combinations of these options — for example, **–fda** or **–frl**.

The following additional keyletters can be used in combination with each other, and with the ones above:

–fe = display element-ID
–fk = display data type of this version (different versions of an element can have different data types)

–k Select only kept elements.

–m Select only modified elements. Can be used with **–d**.

–M Select only missing elements.

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

–o Select only elements with overlaps, which require a merge.

–i Select only stranded elements: members of the default group that no longer have a pathname in this workspace or stream.

–O Override: don't optimize the search for modified files (i.e. don't ignore files whose modification times precede the workspace's update time). The override also enables **stat** to report certain elements as "missing" from the workspace. Having to check all files, regardless of modification time, slows **stat** performance. See *Optimized Search for Modified Files* above.

–p Select only pending elements.

–R Process all the elements in the specified directory tree(s). Use "**.**" to specify the current working directory. You must also specify either **–x** or **–n**; you cannot combine **–R** with other options.

–s <*stream*>
Display the status of the version in the specified stream, not the file in the workspace. See the description of <*element-list*> below.

–x Select only external files and directories.

–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. There is no provision for comment lines in a list-file.

If you use this option, you cannot also specify an <*element-list*>.

<element-list>

One or more element names, separated by whitespace. If you specify a list of elements in this way, you cannot also use the **–l** option.

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**.

Preferences

The following preferences are implemented through environment variables. Setting of environment variables differs among operating systems and shell programs.

ACCUREV_IGNORE_ELEMS

Takes a space-separated list of filename patterns. Causes **stat –x** to ignore external files that match any of the patterns. Note that this preference has no effect on files that are already elements (have been added to the depot with the **add** command).

Example: in the Unix Bourne shell, have **accurev stat –x** ignore temporary files and text-editor backup files:

```
export ACCUREV_IGNORE_ELEMS="*.tmp *.bak"
```

(In general, enclose the pattern list in quotes on Unix systems, but not on Windows systems.) See also *Using the ACCUREV_IGNORE_ELEMS Environment Variable* on page 39 in *AccuRev Technical Notes*.

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
```

See Also

Common Command Options on page 22, **add**, **dir, files**

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, 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 Concepts Manual*

translist — list transactions containing versions that need to be promoted

Usage

```
accurev translist [ -s <stream> ]
```

Description

The **translist** command lists all the transactions — **keep**, **anchor**, **move**, **defunct**, etc. — that created the versions currently in the default group of the specified stream (default: your workspace stream). These are the versions that need to be promoted from this stream to its parent stream.

Some or all of these transactions might also include versions that have already been promoted; if a transaction contains even a single unpromoted version, it's included on this list.

Options

–s <*stream*>

Specify the stream whose versions need to be promoted. If you omit this option, **translist** uses your workspace stream.

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

patchlist, **mergelist, hist**

touch — update the timestamp of a file

Usage

```
accurev touch [ -R ] <file-list>
```

Description

The **touch** command updates the time-last-modified timestamp on one or more files in your workspace. It uses the system clock on your host (the client computer) to set the timestamps.

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.

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**

unarchive — restore version container files that were previously archived

Usage

```
accurev unarchive [ -R ] [ -s <stream> ] [ -t <transaction-range> ]
   [ -i ] [ -E <element-type> ] <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.

The command-line options of the **unarchive** command must 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
  ```

See Also

archive, **reclaim**

undefunct — restore a previously removed element to a workspace

Usage

```
accurev undefunct { <element-list> | -l <list-file> | -e <eid> }
```

Description

The **undefunct** command undoes the effect of the **defunct** command: it restores one or 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

The **undefunct** command fails if both of these conditions hold:

- You are in a workspace where exclusive file locking is in effect (serial-development mode).
- Some other workspace based on the same backing stream already has any of the specified files under active development (**anchor**, **co**, **revert**, **keep**, **move**, **defunct**, **undefunct**).

See *Serial Development through Exclusive File Locking* on page 25 of the *AccuRev Concepts Manual*.

Options

–e <*eid*>
: Operate on the element with the specified element-ID. You can use this option instead of specifying a list of 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. There is no provision for comment lines in a list-file.

  If you use this option, you cannot also specify an <*element-list*>.

<element-list>
: One or more element names, separated by whitespace. If you specify a list of elements in this way, you cannot also use the **–l** option.

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, enabling 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, which prevents all promotions, **incl**/**excl** commands, and **chstream** commands.

Options

–kf ("from" lock) Re-enable promotions *from* the stream.

–kt ("to" lock) Re-enable promotions *to* the stream.

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 [ -i ] [ -t <transaction-number> ] [ -r <reftree> ]
```

Description

The **update** command incorporates into your workspace other people's changes, which they have previously promoted to your workspace's backing stream. In most cases, this has the effect of copying versions from your workspace's backing stream into your workspace.

Namespace-Related Changes vs. Content Changes

update distinguishes between an element's namespace-related changes and its content changes:

- Content changes are the familiar ones handled by all version-control systems: adding a line of code to a Java source file, replacing one image with another in a JPG file, and so on. This kind of change applies to file elements only, not to directory elements.
- Namespace-related changes affect an element's location with the depot's directory hierarchy. AccuRev handles, and distinguishes between, two kinds of namespace-related changes:
  - Renaming of an element — for example, changing the name of file **utils.java** to **gizmo_utils.java**.
  - Moving of an element from its current directory to another directory — for example, moving file **.../cmd_interface/commands.java** into a subdirectory, so that it becomes **.../cmd_interface/utils/commands.java**.

  Namespace-related changes apply both to file elements and to directory elements.

The way in which **update** incorporates changes from the backing stream depends on what kind(s) of changes they are, and also on what changes, if any, you've made to the element in your workspace. The general principle is that changes you've made in your workspace will remain unaffected — the **update** command never "clobbers" your own work.

Here are the details:

- **If you haven't made any changes to the element in your workspace**: the element's status in your workspace will be **(stale)**. An update copies the version in the backing stream to your workspace. Thus, all changes to the element in the backing stream — namespace-related and/or content — get incorporated into your workspace.
- **If you've made only content changes to the element in your workspace**: the element's status in your workspace will be **(overlap)**. One or both of the status flags **(kept)** and **(modified)** will also be present. An update will not change the content of the element, even if the backing stream also contains a content change.

But if the backing stream contains a namespace-related change, an update *does* apply that change to your workspace: the file "jumps" to another location, but its contents remain the same. The file's version-ID and timestamp remain unchanged.

- **If you've made namespace-related (and perhaps content changes) to the element in your workspace**: the element's status in your workspace will be **(overlap)** and **(kept)** — and perhaps **(modified)**, also. An update always leaves the element unchanged.

In some of these situations, the **update** command aborts when it encounters a file to which a content change cannot be applied. See *Refusal to Update* below.

Note that a namespace-related change to a directory element can have "side effects" on other elements. For example, **update** might incorporate the renaming of a directory from **src** to **java_source** into your workspace. This has the side effect of changing the pathname of every element in the subtree under the renamed directory.

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 **files** and **stat** commands. See *Optimized Search for Modified Files* on page 155.

Target Level, Update Level, and Update Time

update causes new values to be recorded for three workspace parameters, the target level, the update level, and the update time.

When an **update** command begins, it sets the workspace's target level to the depot's most recent transaction (or to the transaction you specified with the **–t** option). Then, it attempts to incorporate into the workspace changes that were recorded in the depot as transactions, up to and including the target transaction. As it incorporates the changes, it keeps track by continually incrementing the workspace's actual update level. When **update** completes successfully, the target level and update level are the same.

The target level and update level are displayed by the **show wspaces** command:

```
> accurev show wspaces
Workspace        Storage                  Host       Stream#
...
gizmo_dvt_jjp    C:/wks/gizmo/dvt_jjp     moped      6 56 47 1 0
...
```

In this example, the workspace's target level is transaction #56, and its update level is transaction #47. The discrepancy indicates that the most recent update of this workspace was interrupted before it could be completed.

Note: After a successful **update**, changes made by you and other users will cause the depot's transaction level to increase beyond your workspace's update level. But the update level remains the same until you perform another **update**.

AccuRev also notes the time that the **update** was executed, and records this as the workspace's update time. The update level and update time are *not* equivalent. For example, suppose you perform an **update** just before a 3-day weekend, during which no AccuRev commands are executed. If you perform another **update** when you return to work, the update level remains the same (there were no new transactions in the depot), but the update time is incremented from before-the-weekend to after-the-weekend.

Why would you perform an **update** when there's been no AccuRev activity? Certain AccuRev commands, including **files**, **stat** and **update** itself, use a performance optimization based on the update time. In general, the more recent the update time, the better these commands perform. For more information, see *Optimized Search for Modified Files* and *Using 'update' to Improve 'stat' Performance* on page 155.

Refusal to Update

If you have changed any file in your workspace, and it is not yet a member of the default group for that workspace, **update** refuses to proceed. In this situation, an update would overwrite your changes to the file, which haven't been preserved anywhere in the AccuRev repository. To enable updating of the workspace, you can **anchor** those files to their current version in the workspace stream. Alternatively, you can **keep** your changes or **purge** them.

update's search for "modified non-member files" is, in effect, a **stat –n** command. The value of the ACCUREV_IGNORE_ELEMS environment variable can affect the results of this search; thus, it can affect **update**'s decision on whether to proceed. In particular, if ACCUREV_IGNORE_ELEMS filters out the names of all modified non-member files, **update** will decide to proceed, even though it shouldn't. But those files won't be clobbered — **update** checks each file's status before overwriting it. If it happens to be a modified non-member file, the update terminates immediately with this message:

```
Encountered modified element <pathname>
Resolve with anchor, keep, or purge before running update again.

Warning: Update incomplete.
```

See also *Using the ACCUREV_IGNORE_ELEMS Environment Variable* on page 39 in *AccuRev Technical Notes*.

Updating Sparse Workspaces

> Note: sparse workspace are now deprecated. Use the include/exclude facility instead. See the **incl** reference page.

In a sparse workspace, **update** does not "fill in" new files that were added to the depot since the last update. To get such files, follow the **update** command with a **pop** command:

```
accurev pop -R <top-level-directory-of-workspace>
```

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) 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 61 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 or a series of **co –v** commands. See *Using Checkouts or Merges to Implement a Partial Update of Your Workspace* on page 89.

Options

–i Show which files would be updated, but don't perform the update.

–r <*reftree*>
Specify a reference tree to be updated. (Default: update the current workspace.)

–t <*transaction-number*>
Update the workspace only to the specified transaction. Versions that entered the backing stream after this transaction will not be copied to the workspace.

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

Common Command Options on page 22, **co**, **merge**, **keep**, **purge**, **pop**, **incl**

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). 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 **promote**d or **purge**d the change):

- **keep** (including **keep**s 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 streams that are below the stream specified with **–s**.
  - **–u**: includes streams that are above the stream specified with **–s**.

–fx
: Display results in XML format.

–l <*list-file*>
: Consider only the elements listed in <*list-file*>. This must be a text file, with one element per line. Each element must be specified by a depot-relative pathname. Extra whitespace is not allowed; make sure there are no empty lines and no leading or trailing white space around the filenames. There is no provision for comment lines in a list-file.

  If you use this option, you cannot also specify an <*element-list*>.

<element-list>

One or more element specifications, separated by whitespace. Each element must be specified by a depot-relative pathname. If you specify a list of elements in this way, you cannot also use the **–l** option.

Examples

List all development activity taking place in workspaces backed by stream **tulip_dvt**:

```
> accurev wip -s tulip_dvt

tulip_dvt_jjp
   \.\src\base.h
tulip_dvt_mary
   \.\tools\perl\findtags.pl
tulip_dvt2_jjp
   \.\doc\chap02.doc
   \.\tools\cmdshell\start.sh
   \.\tools\perl_work
   \.\tools\perl_work\reporter.pl
```

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
  kestrel_dvt_jjp
\.\src\brass.h
  kestrel_dvt_jjp
```

See Also

stat, **files**, **incl**, **excl**