Subversion 1.9 for Developers

Essential Concepts 1

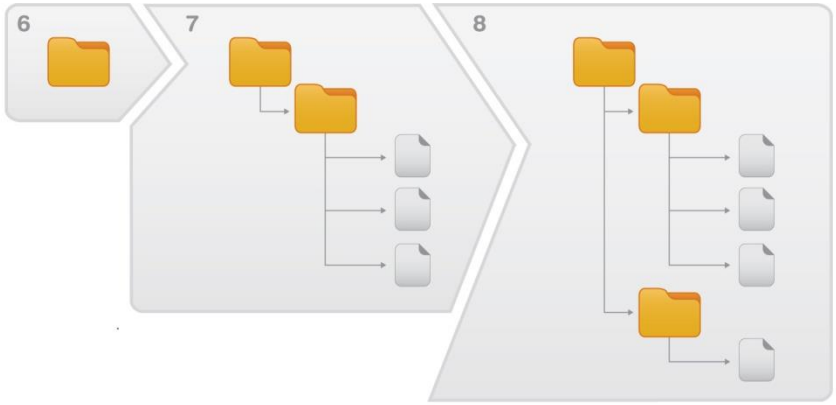
Outline

Typical Usage

- Global revisioning
- Where to get help
- Working copy
- Standard work cycle
- Mixed revisions
- Examine history
- Properties
- Other useful commands

Global revisioning

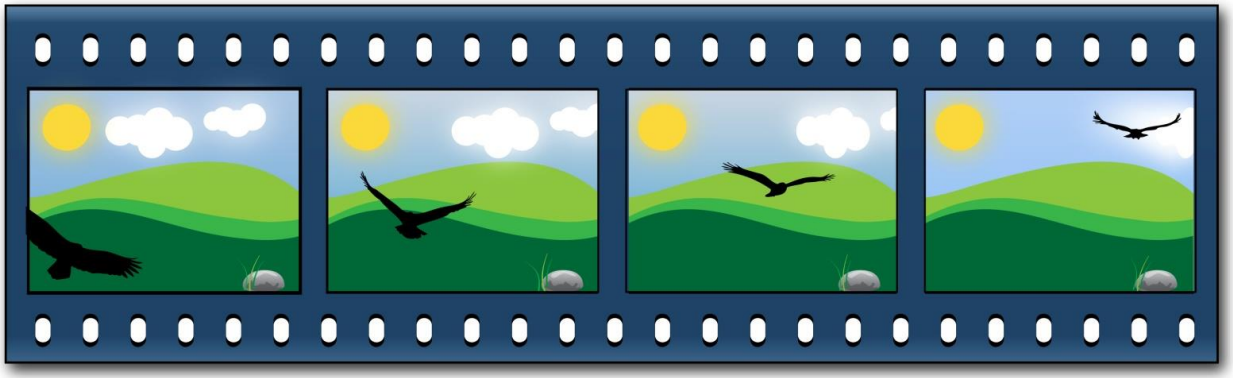
Each revision number:



- Identifies a version tree.
- Has an associated changeset.
- Is per-commit.
  - Not per-file nor per-branch per-file.
  - Commits are atomic (all changes happen or none).

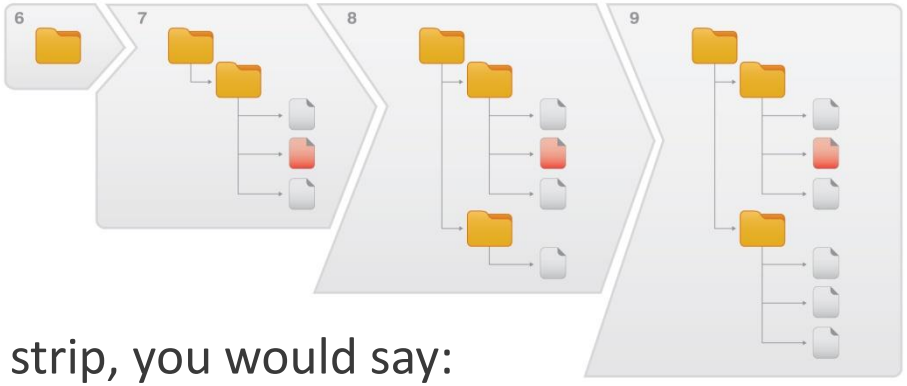
A repository is a series of version trees.

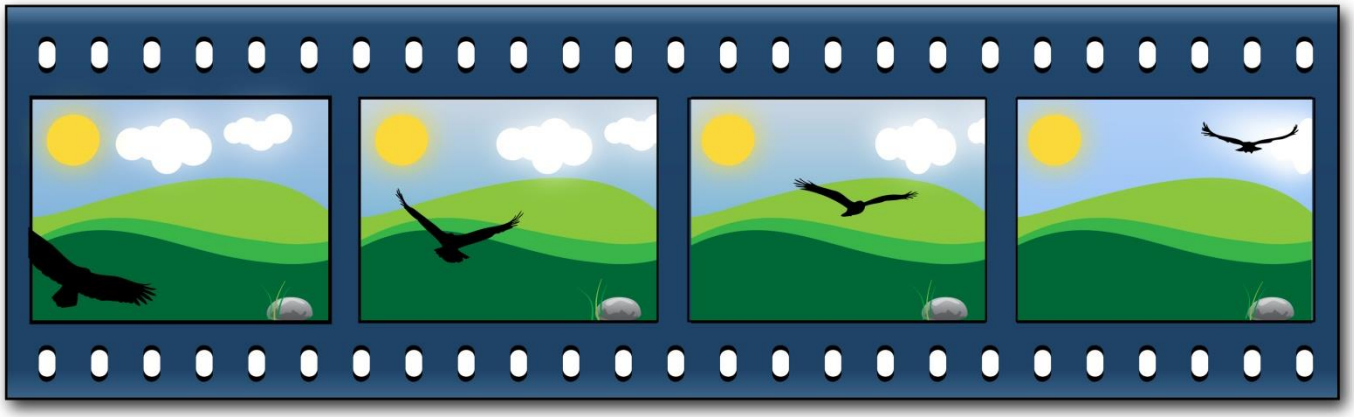
- A repository is the equivalent of a film strip with a revision being the equivalent of a frame in the film.



Global revisioning (Cont'd)

- When talking about versioned objects in Subversion, you would say:
  - *"File red as it appears in revision 7" (or "red in r7").*
- But you would not say:
  - *"version 7 of file red".*
- Continuing with the analogy of a film strip, you would say:
  - *"The bird as it appears in frame 7".*





Global revisioning (Cont'd)

- Subversion keeps track of two revision numbers for each checked out object and can report that information:
  - What revision is presented in the working copy.
  - What revision last changed the object.
- The `info` command shows the two revision numbers:
  - Our working copy has the file red as a part of the revision 7 tree, but
  - Red itself has not changed since revision 3.

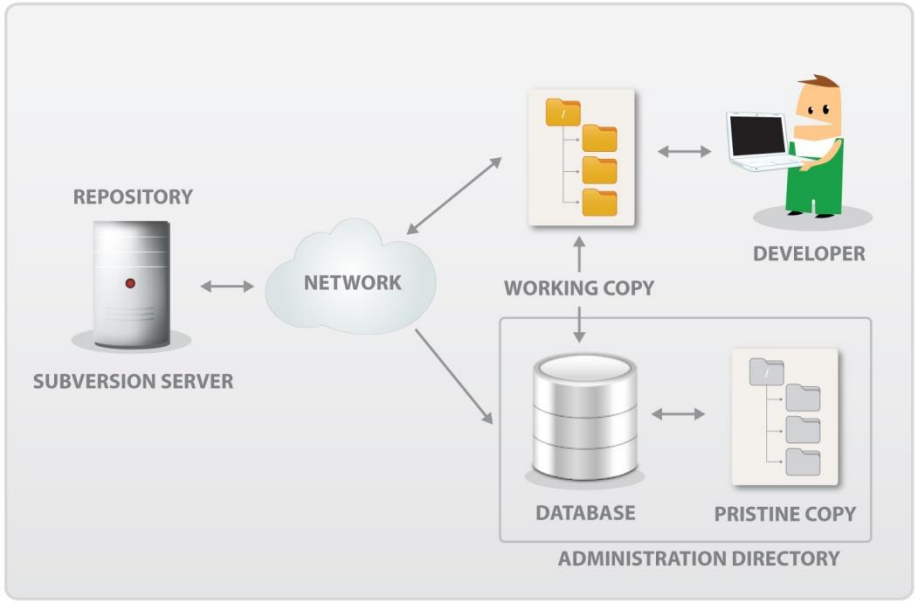
```
$ svn info red
    Path: red
    Name: red
    URL: http://repos.com/proj/trunk/red
    ...
    Revision: 7
    Node Kind: file
    ...
    Last Changed Rev: 3
    ...
    $
```

Where to get help

- Integrated help:
  - Command line: `svn help <subcommand>`
  - Help menu in all mature clients
- Subversion books:
  - http://svnbook.red-bean.com/
  - Chapter 9 provides complete reference
- Quick reference card:
  - http://www.open.collab.net/community/subversion/articles/SvnQuickReferenceCard.html
- Discussion forums:
  - http://forums.open.collab.net
- Users' mailing list:
  - users@subversion.apache.org

Working copy

- A working copy comprises:
  - A copy of a revision of the requested data set.
  - Another copy of the same revision of the data set, the so-called “pristine copy” (i.e., base version), to enable sending only deltas across the wire and to support offline operations.
  - Administrative data (e.g., revisions in working copy, locks, properties, etc.) located in a SQLite database.



Creating a working copy

- Use checkout to create your working copy.

```
svn checkout http://svn.collab.net/repos/svn/trunk
```

- You can checkout:
  - The trunk
  - A branch
  - A tag
  - A sub-tree of any of the three previous targets.
- You can, and likely will, have multiple working copies on your system to alternate working on different tasks and lines of development.
- Use `info` to list the repository and revision a working copy points to.

Sparse checkouts

- Sparse checkouts can control working copy population when you:
  - Want to create working copy with just a selected subset of the full tree.
  - Want to pull in sub trees as you need them.
  - Desire to reduce the chances of touching files you are not supposed to touch.
- Sparse checkout definitions are stored client-side, per-working-copy.
- Examples:
  - Checkout an empty tree.

```
svn co --depth=empty http://svn.collab.net/repos/svn/branches/1.4.x
```

  - Need only some top-level files.

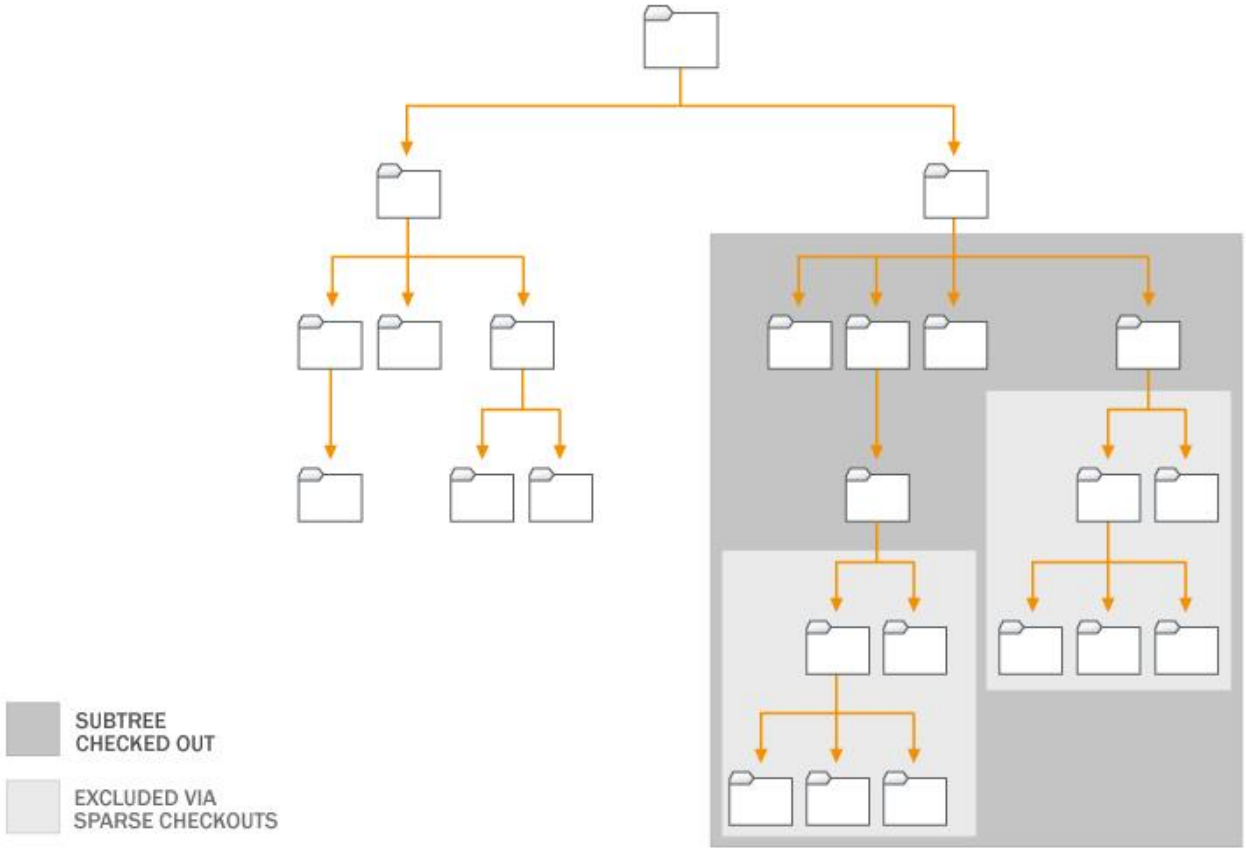
```
cd 1.4.x
svn update STATUS CHANGES README
```

  - Want just one or two components/projects in a large repository.

```
svn update component1 component5
```

Sparse checkouts (Cont'd)

- Checking out only a sub tree and sparse checkouts complement each other in enabling a user to limit what is in their working copy.
  - You can use different checkout levels to get just the directories you want.



Standard work cycle

1. Update your working copy

- Use `update` to update to the latest changes in the repository.
  - A working copy, folder or file can be the scope of an update.

```
cd my-working-copy
svn update
```

BEST PRACTICE

Update from the top of your working copy.

- Update reports status for each changed path indicating what was:
  - Updated (U)
  - Added (A)
  - Deleted (D)
  - Replaced (R)
  - Merged (G)
  - Conflicting (C)

2. Make your changes

- If working using the lock-modify-unlock approach, use lock to lock a file before changing it.
  - Locks are mainly a communication tool.
  - Locks can be released by anyone (broken or stolen).
    - Has to be done consciously.
    - Action is visible and traceable by others.

```
svn lock –m “urgent fix” image.png
```

- Make your changes.
  - Change file content: use your editor of choice.
  - Change structure: use an IDE or Subversion clients to add, copy, delete, and move/rename paths.

- A common pitfall when using command-line or clients like TortoiseSVN is to use local file system commands to change the structure. These will not be reflected in Subversion.

2. Make your changes (Cont'd)

- For structural changes use:
  - `add` – add new paths

```
svn add foo.c
```

  - `copy` – copy paths

```
svn copy foo.c bar.c
```

  - `delete` – delete paths (only deletes them moving forward in history)
    - You can keep a local copy of what is deleted with the keep-local option.

```
svn delete foo.c
```

  - `move/rename` – move paths (rename is an equivalent command)
    - A move or rename is executed and reported as a copy with history plus a delete of the original path.
    - It is recursive: when renaming a directory, `status` will show a copy with history of the directory and separate deletes for every path in the sub-tree.
    - These are tracked locally as a first class move, but not stored as such.

```
svn move bar.c ../module2/bar.c
svn rename bar.c baz.c
```

3. Examine your changes

- Use `status` to list an overview of changes in your working copy.
  - Report defaults to showing local modifications (including moves/renames).
  - Use -u flag to also show remote modifications and anticipate conflicts (i.e., update preview).

```
cd my-working-copy
svn status -u
```

- Use `diff` to examine exactly what has changed.
  - Defaults against the “pristine” version (offline operation).

```
svn diff foo.c
```

  - Works between any two revisions in repository.
  - Can also work between a revision in the repository and your working copy version.
  - Can also compare arbitrary files (see --old and --new parameters)
  - Plugging in external, format-specific diff tools is supported.

3. Examine your changes (Cont'd)

Example output of `status -u` for reference:

```
L     some_dir              # svn left a lock in the .svn area of some_dir
M     bar.c                 # the content in bar.c has local modifications
M     baz.c                 # baz.c has property but no content modifications
X     3rd_party             # dir is part of an externals definition
?     foo.o                 # svn doesn't manage foo.o
!     some_dir              # svn manages this, but it's missing or incomplete
~     qux                   # versioned as file/dir/link, but type has changed
I     .screenrc             # svn doesn't manage this, and is set to ignore it
A +   moved_dir             # added with history of where it came from
M +   moved_dir/README      # added with history and has local modifications
D     stuff/fish.c          # file is scheduled for deletion
A     stuff/loot/bloo.h     # file is scheduled for addition
C     stuff/loot/lump.c     # file has textual conflicts from an update
C     stuff/loot/glub.c     # file has property conflicts from an update
R     xyz.c                 # file is scheduled for replacement
S     stuff/squawk          # file or dir has been switched to a branch
K     dog.jpg               # file is locked locally; lock-token present
O     cat.jpg               # file is locked in the repository by other user
B     bird.jpg              # file is locked locally, but lock has been broken
T     fish.jpg              # file is locked locally, but lock has been stolen
```

4. Bring in others' changes

- Use `update` to bring in changes made and committed by others.

```
cd my-working-copy
svn update
```

  - Automatically merges committed changes made by others that do not overlap with your changes.
    - Informs you of all changes made to your working copy.
  - Raises a conflict for committed changes that overlap with your local changes.

BEST PRACTICE

If an update makes ANY changes to the working copy, it is recommended that you consider whether you need to rebuild and retest before committing your changes.

4. Resolve conflicts, if any

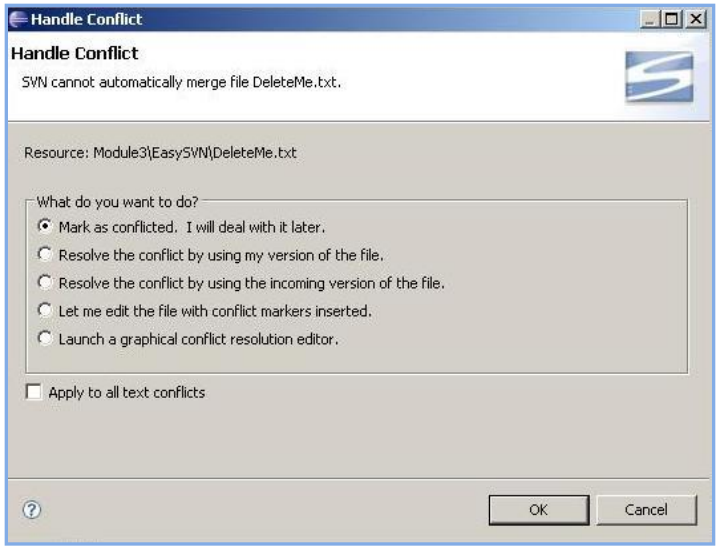
- For each path in conflict, Subversion will:
  - Signal the conflict by printing a “C” in the summary output.
  - Place a flag on the path in conflict.
    - A path flagged as conflicted cannot be committed until explicitly resolved.
  - Insert conflict markers in a file if it is in a mergeable format, e.g.:

```
<<<<<<< .mine
"Hello hello, world!"
||||||| .r1
"Hello, world!"
=======
"Hello, ${name}!"
>>>>>>> .r2
```

  - Write three temporary files in your working copy:
    - filename.mine – the contents of the file in your working copy before the update.
    - filename.rOLDREV – the contents of the file from your last update or checkout (the “pristine version”).
    - filename.rNEWREV – the contents of the file from the latest revision in the repository.

4. Resolve conflicts, if any (Cont'd)

- Resolve conflicts, either by:
  - Leaving it unresolved and dealing with it later.
  - Using either your version or the revision from the repository.
  - Editing it manually.
  - Launching a graphical conflict resolution editor.



- Use `resolve` to acknowledge resolution.
  - Use after resolving the conflicts or,
  - Use interactively after initially postponing resolution to implement resolutions or,
  - Use the `accept` option to choose a specific version of a file (base, mine or theirs) for combining resolution with acknowledgement in one operation.
    - Resolves binary file conflicts easily.

```
svn resolve DeleteMe.txt
```

4. Interactive conflict resolution

- With interactive conflict resolution:
  - Quicker and easier for you to resolve conflicts (executes immediately after the operation completes or you can execute it via the resolve command at a later point in time).
  - Available with `update`, `switch`, and `merge` commands.

```
svn update

U contrib/client-side/svnmerge/svnmerge_test.py

Conflict discovered in 'contrib/client-side/svnmerge/svnmerge.py'.

Select: (p) postpone, (df) diff-full, (e) edit,
        (mc) mine-conflict, (tc) theirs-conflict,
        (s) show all options:
```

  - Able to pre-specify directives like “always use the version from my merge source”.

More options available if you type “s” for show more options.

5. Commit your changes

- Use `commit` to commit your changes to the repository.
  - Commit automatically releases locks unless explicitly told not to.

```
cd my-working-copy
svn commit –m “fixed bug 123”
```

BEST PRACTICE

When committing:

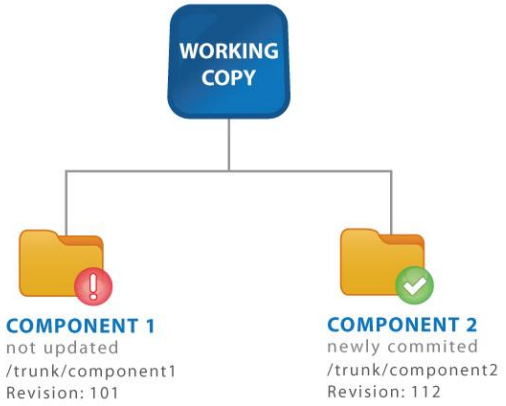
- Execute a build and unit test to ensure your commit will not break anything.
- Commit exactly one logical unit of work at a time.
- Concise, descriptive log messages should be used, e.g.:

```
* lib/twirl.c fixed problem xy #bug-id.

* src/foo.c: reverted r1234 ‘fixed xyz’ due to new regressions.

* src/main.c: added new library package (version)
```

Standard work cycle

Mixed revisions

- A working copy can contain mixed (i.e., multiple) revisions.
  - This can be a result of:
    - Selective commits: only paths touched by commit are at HEAD, rest is at the revision last checked out or updated to.
    - Selective updates (e.g., done on a sub-directory or individual files).
  - If the mixed revisions are not desired, an update from the tree root (i.e., top of the working copy) brings all revisions up to date (for these cases).
  - Example uses for a mixed revision working copy:
    - Keep most of tree at a stable revision, but put a specific sub-tree (e.g. component) at the bleeding edge of development.
    - Backdate selected portions of tree for regression testing.
  - Limitations of a mixed revision working copy:
    - You can only delete an up-to-date path.
    - You can only commit property changes to an up-to-date path.



Examine history

Command	Description
`log`	List log messages with date, author, and log message information attached to the revisions along with the ability to see the paths changed by each revision (i.e., changeset) • For a file, revisions where the file changed • For a folder, revisions where something in the tree changed
`diff`	Show specific details of how a file or structure changed over time
`cat`	Retrieve any file as it existed as of a particular revision number and display it on your screen
`list`	Show the list of files and directories in a directory for any given revision
`blame`	Show a text file with the last changed revision and author information in-line (sometimes called annotate)

Log

- `svn log`

```
------------------------------------------------------------------------
r200 | jrandom | 2007-07-15 10:51:34 -0700 (Sun, 15 Jul 2007) | 10 lines

Add new speed and radius options for twirling batons.
...
------------------------------------------------------------------------
r199 | kathym | 2007-07-15 10:51:34 -0700 (Sun, 15 Jul 2007) | 3 lines
...
```

- `svn log -v` (shows paths changed)

```
------------------------------------------------------------------------
r200 | jrandom | 2007-07-15 10:51:34 -0700 (Sun, 15 Jul 2007) | 10 lines
Changed paths:
  M /trunk/twirl.c
  D /trunk/degrees.c
  A /trunk/conversion-tables.c (from /trunk/degrees.c:130)
Add new speed and radius options for twirling batons.
...
```

Log (Cont'd)

- `svn log --search Bob --search-and "Jun 2012"`

  Show revision data for revisions where the author (or log message text or list of changes paths) matches "Bob" and the commit date is in June 2012.

- `svn log -r193`

  Show just the revision data for r193.

- `svn log -r193:268`

  Show the revision data for r193 through r268 (inclusive).

- `svn log https://example.com/svn/trunk/path`

  Show logs for a repository URL (no working copy required).

- `svn log URL path1 path2 ...`

  Show only revisions in which at least one of the named files or directories was changed.

- There are many more log options, see '*`svn help log`*', and http://svnbook.red-bean.com/nightly/en/svn.ref.svn.c.log.html.

Diff

- `svn diff` (shows local, uncommitted changes)

```
Index: lib/twirl.c
===================================================
--- lib/twirl.c    (revision 199)
+++ lib/twirl.c    (working copy)
@@ -43,7 +43,8 @@
            (screen_context,
             convert_degrees(&new_x,
                             &new_y,
                             degrees,
+                            speed,
+                            radius,
                             baton,
                             rotation_func)

Index: www/manual.html
===================================================
--- www/manual.html  (revision 199)
+++ www/manual.html  (working copy)
@@ -3057,5 +3057,27

  <div id="features" title="features">
+ <p>You can control the speed and radius of the baton by...
```

Diff (Cont'd)

- `svn diff --old foo.c --new bar.c`

  Compare arbitrary files with one another.

- `svn diff -r190:199`

  Show what changed under for this path between r190 and r199.

- `svn diff -r190:191` or `svn diff -c191`

  Show what changed under for this path in exactly r191.

- `svn diff -r190:199 https://example.com/svn/trunk/`

  Show what changed between r190 and r199 for given URL (no working copy required).

- Many more diff options, see '*`svn help diff`*', and http://svnbook.red-bean.com/nightly/en/svn.ref.svn.c.diff.html.

Blame

- `blame` or `annotate` gives a line-by-line annotation of an ascii/text file, showing when each line was last changed and by whom.
- `svn blame twirl.c`

```
18946   patsmith   /* Load ADM_ACCESS's entries file. */
18471    jrandom   entries_read(&entries, access));
18471    jrandom   svn_wc_entries_read(&nohidden, access);
 1532      freda
 1532      freda   /* Ensure that NAME is valid. */
   29     kathym   if (name == NULL)
 2024    jrandom     name = ENTRY_THIS_DIR;
 1532      freda
   29     kathym   if (entry_modify_schedule)
   29     kathym     {
 1532      freda       entry_t *entry_before, *entry_after;
21728   dionisos       int orig_modify_flags = modify_flags;
 9882    navarya       schedule_t schedule = entry->schedule;
```

Properties

- Properties are meta data, key-value tuples.
  - Subversion's reserved properties all start with "`svn:`".
  - Your organization can define any properties it wants outside of that namespace.
  - The auto-props configuration feature maps file name patterns to versioned properties to support them being automatically applied to new files.
- There are two kinds of properties.
  - **Versioned properties** ("props") are applied to a versioned object (i.e., a file or directory). Changes to these properties are versioned.
  - **Revision properties** ("revprops") are applied to a revision. They are not versioned – think of them as notes pinned to revision numbers.
- Subversion commands for properties.
  - `proplist`, `propget`, `propset`, `propdel`, `propedit`

Predefined versioned properties

Property	Description
`svn:executable`	Executable indicator for a file (for Unix/Linux system support)
`svn:mime-type`	MIME classification
`svn:ignore`	File patterns that Subversion should ignore in this directory
`svn:keywords`	Keyword (specific ones) expansion/substitution in files
`svn:eol-style`	Handling of end-of-line markers
`svn:needs-lock`	Indicator that the file needs to be locked before modifying
`svn:externals`	Working copy should be assembled as an aggregate, built from checkouts of different locations (branches, tags, etc), some of which may come from different repositories
`svn:auto-props`	Repository-dictated properties (and values) to automatically apply as file matching patterns are satisfied by new files (via add or import)
`svn:global-ignores`	Repository-dictated patterns Subversion should ignore as potential new paths
`svn:mergeinfo`	A record of merge data
`svn:special`	Indicator that this is a symbolic link (supported only on Unix/Linux servers) or some future special type of file

Example: file locking

- Let's assume we have an existing binary file and we want to require the use of the lock-based approach on that file:
  - Set the `svn:needs-lock` property on a file to make the file read-only (unless you already have the lock for it on that path which would give write access).

```
svn propset svn:needs-lock "*" artwork.png
svn commit -m "Advise locking." artwork.png
```

  - File is read-only until you obtain a lock.

```
svn lock -m "Retouching colors." artwork.png
```

  - And the lock is released upon a commit or an unlock.

```
svn commit -m "Finished retouching." artwork.png
     -OR-
svn unlock artwork.png
```

Predefined revision properties

Revision property	Description
`svn:log`	Log (commit) message for a particular revision
`svn:author`	Username of the creator of the revision (committer)
`svn:date`	Date and time stamp of when the transaction was executed

Clean up working copy

- If a Subversion client is killed in the middle of an operation, the next time you launch Subversion, you may be prompted to use the `cleanup` command to complete the previously started operation.
  - A working copy contains a journaling system which `cleanup` uses to complete previously interrupted operation.
  - If the client tells you part of your working copy is locked, this refers to temporary locks used locally by the client to maintain working copy integrity and is not related to the locks discussed earlier.
  - `cleanup` almost always works but not in every situation. You may need to create a new working copy and move your changes to it.
    - Note that the issues that created the need for this operation in no way compromised the changes you made locally – i.e., there is no risk of corruption.
- This command may also fix client icons if they are not reflecting the true status of your working copy (there isn't a downside to executing this command).

Revert local changes

- Use `revert` to discard your local changes (changes not committed).
  - Reverts changes both to individual files and to a tree structure.
  - Executed as an offline operation (uses the “pristine” copy).

NOTE

It really discards! – if you want a copy of your changes, make a copy before you revert.

- Example:
  - Suppose you mistakenly removed a file from version control.

```
$ svn status README
    README
$ svn delete README
D   README
$ svn revert README
    Reverted 'README'
$ svn status README
    README
```

Thank You

CollabNet

About CollabNet

CollabNet is a leading provider of Enterprise Cloud Development and Agile ALM products and services for software-driven organizations. With more than 10,000 global customers, the company provides a suite of platforms and services to address three major trends disrupting the software industry: Agile, DevOps and hybrid cloud development. Its CloudForge™ development-Platform-as-a-Service (dPaaS) enables cloud development through a flexible platform that is team friendly, enterprise ready and integrated to support leading third party tools. The CollabNet TeamForge® ALM, ScrumWorks® Pro project management and SubversionEdge source code management platforms can be deployed separately or together, in the cloud or on-premise. CollabNet complements its technical offerings with industry leading consulting and training services for Agile and cloud development transformations. Many CollabNet customers improve productivity by as much as 70 percent, while reducing costs by 80 percent.

For more information, please visit www.collab.net.

CollabNet, Inc.
8000 Marina Blvd., Suite 600
Brisbane, CA 94005

www.collab.net

+1-650-228-2500
+1-888-778-9793
blogs.collab.net
twitter.com/collabnet
www.facebook.com/collabnet
www.linkedin.com/company/collabnet-inc