Next Generation Software Configuration Management with Subversion

Agenda

- Why Use Subversion?
- Basic Usage
- Comparison with CVS
- The Cheap Copy
- Additional Tools
- Future Directions
- Original source material: Greg Stein, http://www.lyra.org/greg/presentations/

More Agenda

- Ask questions any time!

Why Use Subversion?

Background

- Version control system
- Goal: displace CVS
  - Start by matching CVS’ feature set
  - Fix concepts/commands that are broken
  - Surpass CVS
- Open Source
- Written from scratch

Developer Benefits

- Standard version control benefits
  - Safety, repeatability
  - Manage multiple lines of development
  - and more (beyond the scope of this tutorial)
- Secure, efficient remote operation
- Transacted operation (“atomic”)
- Simplified branching, tagging, and directory organization
- Tool integration

Administrator Benefits

- Secure, standard authentication mechanisms
- Hot backup capability
- Scalable solution
- Builds on an existing, well-known server (Apache)
- Easy integration with an existing network security policy

Business Benefits

- Developer productivity
- Smoother workflow with partners
- Supports many development models
- Open source
  - Reduces costs
  - Avoids vendor lock-in
  - Increased interoperability and integration

Why was it made Open Source?

- If it is great, then it could replace CVS
  - Widespread usage would establish it as a legitimate tool
- More and better clients are possible
- Peer review and testing
  - Broad-base testing is very important for a version control tool
- Community feedback and contributions

Basic Usage

Simple Usage Model

- Four basic steps
  - Create a “working copy” on your local disk
  - Make changes
  - Potentially merge changes from server
  - Commit your changes to the server
- Client-side editing
- “Unreserved” model – no locking

Subversion Clients

- Different clients enables matching of users’ needs
- Command line (all platforms)
- TortoiseSVN (Windows shell extension)
- IDE support (e.g. Eclipse, Xcode, DevStudio)
- Various WebDAV clients
- *many more…*

Administrator Usage

- Set up the server
  - Mapping of URLs to filesystem locations
  - Harder: authentication and authorization
- Create repositories
- Perform backups
- Monitoring

Example: Basic Usage

```
$ cd $HOME
$ mkdir repos src
$ svnadmin create repos/test
$ svn co file://$HOME/repos/test src/test
Checked out revision 0.
$ cd src/test
$ vi README.txt
$ svn add README.txt
A         README.txt
$ svn commit -m 'Add a simple README.‘
Adding         README.txt
Transmitting file data .
Committed revision 1.
$
```

Comparison with CVS

Other Version Control Systems

- Open Source
  - CVS
  - Arch
  - Less popular: RCS, OpenCM, Aegis, …
- Commerical
  - Perforce
  - ClearCase
  - BitKeeper
  - PVCS
  - SourceSafe
  - *many others…*

Subversion vs CVS

- Most CVS features
  - Some differences to improve the system
- Improvements on many CVS features
  - Atomic commits
  - Better binary file handling
  - Designed for the network
  - Direct repository operation
- Going beyond CVS
  - Metadata
  - Directory versioning
  - Layered library design

Feature Comparison (1 of 3)

- Subversion “feels familiar” to CVS users
- Most commands are the same: checkout, add, commit, etc
  - ***`svn command options… files…`***
- Some changes to options
  - Unified options, rather than global and command-specific
  - Long option names are provided
  - Better command-line help
- Omitted edit/watch system and “**`cvs history`**”

Feature Comparison (2 of 3)

- Subversion has additional commands
  - copy, move
  - merge
  - resolved
  - mkdir
  - propset, propget, proplist, propdel, propedit
  - revert
  - switch
  - info
  - cat, list

details on later slides…

Feature Comparison (3 of 3)

- Some things are done differently
  - Revision numbering
  - Status
  - Branching
  - Tagging
  - Authentication (“**`cvs login`**”)
  - Modules
  - Keywords

Difference: Revision Numbering

- Global revision number rather than per-file
- Allows you to talk about “revision 2524”
- Unique identifier for a state of your project
  - Simple way to tag
- Each revision corresponds to a single commit
  - Contains author, log message, and date

Difference: Status

- "`cvs status`" is not very useful
  - Provides status of working copy, and what updates are needed
  - Very verbose (**nine** lines per change) – hard to "see at a glance"
  - Typical workaround: "`cvs update -n`"
  - Both `status` and `update` contact the server
- "`svn status`" provides short, concise feedback
  - One line per local modification
  - Offline operation, by default
  - Option to contact server to look for updates

Difference: Branching and Tagging

- Based on Subversion’s “cheap copies”

Detailed discussion later…

Difference: Authentication

- CVS uses a custom authentication mechanism
  - Part of CVS's custom (non-standard) protocol
  - "`I LOVE YOU`" or "`I HATE YOU`"
  - pserver sends passwords in the clear
- Alternate CVS authentication schemes
  - kserver, gserver
  - SSH tunneling
- Subversion uses HTTP as its protocol
  - Integrates with existing authentication systems
  - Standardized!
- Can also be tunneled through SSH

Difference: Modules

- Modules are used to create composite working copies
- CVS modules
  - Live in CVSROOT
    - The "**`modules`**" file
    - Extra work to allow users to alter module definitions
  - Only apply to checkout
    - Changes are not detected during "**`cvs update`**"
- Subversion modules
  - Directory property ("**`svn:externals`**")
    - Users can define them, edit them, inspect them
    - Attach to any directory
  - Versioned, as with any property
    - Changes are detected during "**`svn update`**"

Difference: Keywords

- CVS keywords are automatically expanded
  - User must explicitly disable this behavior
  - Risk of destroying a binary file
- Subversion keywords are optionally expanded
  - User must proactively enable keyword expansion
  - The user states the set of keywords to expand (some or all)
  - The behavior is controlled by a property: **`svn:keywords`**

Various Improvements

- Atomic commits
  - CVS can commit one file, fail on the second
  - Subversion commits **all** changes, or nothing
- Binary file handling
  - Subversion uses MIME types
  - Binary deltas
- Newline and keyword handling is improved
  - Subversion does not munge your files until you tell it to
- Many operations can be used offline

Improved: Network Operation

- Subversion was designed for the network
- WebDAV/DeltaV support planned from day one
- Custom “svn” protocol came later
  - The repository access system had been designed to make this easy
- Binary diffs in both directions on the network
- CVS had network support “bolted on”
  - Two code paths to maintain
  - Authentication poorly integrated

Improved: Direct Repository Operations

- In some cases, it is useful to avoid a working copy
  - Automated scripts
  - Some operations are handled more efficiently by the server
- CVS has a few operations: **`rtag`**, **`rlog`**, **`rdiff`**, **`rannotate`**
- Most Subversion commands can operate directly
  - Property operations on files, directories, and revisions
  - Modify operations: **`copy`**, **`delete`**, **`mkdir`**, **`move`**
  - Read operations: **`blame`**, **`cat`**, **`diff`**, **`list`**, **`log`**

New: Metadata

- Any file or directory can store properties
- Properties are name/value pairs
- Some standard properties exist
  - **`svn:ignore`**
  - **`svn:mime-type`**
  - **`svn:eol-style`**
  - etc.
- User-defined properties are allowed
- Property values can be text or binary
- Revisions also have properties
  - Standard properties for author, date, and the log message

New: Directory Versioning

- Directory structures are versioned items
- Deletes and moves of files and subdirectories are recorded
- Copy sources are remembered
- Copies are cheap

New: Layered Library Design

- Many levels for interaction
  - High-level client
  - Repository access (local, remote, pipe, etc)
  - Direct access to the storage
- Enables scripting
- Clients can share a lot of code
  - The command-line client is a small application built on top of the high-level client library
  - GUI clients can also use the high-level library
- Library-based approach enables third-parties

The Cheap Copy

Subversion’s “Cheap Copies”

- Copying a file or directory in Subversion is “cheap”
  - Very little extra space required
  - Fast, constant-time operation
- Defines Subversion’s approach to several problems
  - Branching
  - Tagging (aka labels)
  - Development methodologies (really, branch usage)
- Flexible repository layout
  - No worries about “getting it wrong” – it can always be fixed
  - Refactoring is much easier

Branches and Tags

- Branches are just copies of the main trunk
  - Make changes in the copy
  - Merge changes to or from the main trunk
- Tags are copies which are never changed
  - Simple way to apply a name
  - Might not even be necessary if you simply record the global revision number that built your product
- Vast improvement over CVS

Example Repository Layout

```
http://svn.example.com/repos/project/
    trunk/
        source/
        docs/
        buildtools/
    branches/
        issue-1003/
        brian/
    tags/
        alpha-1/
        1.0.0/
        1.0.1/
```

Just an example – you are free to structure the repository in whatever way fits your project’s needs and goals

Example Tag Operation

- Use a direct repository operation for efficiency

```
$ svn copy -m 'Release 1.0.3' \
    http://svn.example.com/repos/project/trunk \
    http://svn.example.com/repos/project/tags/1.0.3

Committed revision 1724.
$
```

Additional Tools

Additional Tools

- **cvs2svn**
- **ViewCVS** (misnomer – it also handles Subversion)
- Hook scripts
  - Send commit emails
  - Simple ACL support
  - Simple reserved checkouts
  - Repository backup
- Libraries, scripting, svnlook

The Best Tool

- Physical, rather than a software tool
- "*Version Control With Subversion*", by C. Michael Pilato, Ben Collins-Sussman, and Brian W. Fitzpatrick
- Published by O'Reilly & Associates
- Also available under a Creative Commons license
  - See http://svnbook.red-bean.com/
  - Comes as part of many Subversion distributions

- See also, "*Practical Subversion*", by Garrett Rooney

Other References

- Subversion’s home: http://subversion.tigris.org/
  - Many links, documents, downloads, and more
- User’s mailing list: users@subversion.tigris.org
  - Large, active community to help users

Future Directions

Subversion 1.1

- Internationalization
  - Localized for: de, es, ja, nb, pl, sv
- Operations follow ancestry
- Versioning of symbolic links
- Additional repository format

After 1.1

- Reserved checkouts
- Merge/branch tracking
- Additional localizations (ongoing)
- Remote management of access control
- Increased WebDAV interoperability
- Relational database repository option
- Pluggable client-side diff/merge tools

Subversion 2.0?

- Version numbers are based on compatibility rather than features
- Development team works very hard to retain compatibility, so 2.0 might not happen
- Subversion 1.x are feature releases

Other Areas of Expansion

- More clients
- More IDE integrations
- Systems which embed/use Subversion

Final Questions and Answers

Detailed Usage

Details: repositories

- Subversion uses URLs for repository locations
- **`http://svn.collab.net/repos/svn/`** is the actual URL for Subversion itself
- Web browsers can view the “head”
  - Use a tool like ViewCVS to browse old revisions, changes, etc
- “file” URLs are also allowed for local access
  - Example: **`file:///home/brian/repos/testing/`**
- “svn” URLs for the custom Subversion protocol
  - Example: **`svn://svn.example.com/project1/`**

Details: Getting Help

- Subversion recognizes **`--help`**, **`-h`**, **`-?`**, and "**`svn help`**"
- Without a subcommand, a list of subcommands is provided
- With a subcommand, that subcommand's help is provided
- In general, help is printed when arguments are incorrect
- "**`svn --version`**" to print version information

Details: checkout

- Creates a local working copy

```
$ svn checkout http://svn.example.com/repos/project/trunk
A  trunk/file1
A  trunk/file2
A  trunk/subdir/file3
A  trunk/subdir/file4
Checked out revision 5.
$ cd trunk
$ ls -aF
./  ../  .svn/  file1  file2  subdir/
$
```

Details: commit

- Commit changes to the repository

```
$ vi file1
$ svn commit -m "changed file1"
Sending        file1
Transmitting file data .
Committed revision 6.
$
```

Details: add

- Add new files and directories

```
$ touch file5
$ mkdir subdir2
$ svn add file5 subdir2
A         file5
A         subdir2
$ svn commit -m "added items"
Adding         file5
Adding         subdir2
Transmitting file data .
Committed revision 7.
$
```

Details: mkdir

- Simplify directory creation

```
$ svn mkdir subdir3
A         subdir3
$ svn commit -m "added subdir3"
Adding         subdir3

Committed revision 8.
$
```

Details: mkdir <URL>

- Quickly sets up a new repository directory

```
$ svn mkdir http://svn.example.com/repos/project/branches \
    -m "create branches area"

Committed revision 9.
$
```

Details: delete

- Delete files and directories

```
$ svn delete file5 subdir3
D file5
D subdir3
$ svn commit -m "deleted items"
Deleting       file5
Deleting       subdir3

Committed revision 10.
$
```

Details: delete <URL>

- Delete items directly from the repository
  - Great for removing obsolete tags or branches

```
$ svn delete \
    http://svn.example.com/repos/project/branches/issue-10 \
    -m "delete unused branch"

Committed revision 11.
$
```

Details: update

- Retrieve changes made by other users

```
$ svn update
U   ./file2
A   ./newfile
Updated to revision 12.
$
```

The above example assumes that another user has created revisions 11 and 12. We update the working copy from revision 10 to 12.

Details: status

- Shows changes to the working copy

```
$ svn status
M      ./file2
M      ./moved-dir/file3
$ svn status –u
M       *        12    ./file2
M                12    ./moved-dir/file3
Status against revision:     13
$
```

Details: copy

- Copy files and directories
  - Source and destination can be working copies and/or direct repository references

```
$ svn copy file1 file6
$ svn commit -m "made a copy"
Adding         file6

Committed revision 14.
$
```

Note: Subversion remembers that **`file6`** came from **`file1`**.

Details: copy <URL> <URL>

- Example provided earlier (to “tag” a release)
- The URL-to-URL form is most often used for creating branches and tags
- Fast, constant time: very little network usage, and the server has very little work
- Cheap enough to tag an hourly or daily build
  - Probably want to delete these tags, or move to subdirectories, to avoid overwhelming humans with large numbers of tags

Details: move

- Move files and directories
  - The source and destination must both be working copy references, or they must both be URLs

```
$ svn move subdir moved-dir
A         moved-dir
D   subdir/file3
D   subdir/file4
D   subdir
$ svn commit -m "moved a dir"
Adding         moved-dir
Deleting       subdir

Committed revision 15.
$
```

Note: Subversion remembers that **`moved-dir`** came from **`subdir`**.

Details: diff

- Shows changes to the working copy
- Very fast, since Subversion has a local copy of the original

```
$ svn diff
Index: ./file2
===================================================================
--- ./file2
+++ ./file2     Tue Jul 11 17:41:15 2002
@@ -1,2 +1,3 @@
 foo
 bar
+baz
$
```

Details: log

- Shows changes that have been committed

```
$ svn log file1
----------------------------------------
rev 2:  gstein | Tue, 12 Jul 2002 15:53:56 -0700 | 1 line

Changed file1
----------------------------------------
rev 1:  gstein | Tue, 12 Jul 2002 13:30:03 -0700 | 1 line

Initial checkin
----------------------------------------
$
```

Details: blame

- Displays who edited each line of a file, and in which revision
- Useful to answer questions like, “who wrote this function?”

```
$ svn blame hello.sh
   341      gstein #!/bin/sh
   341      gstein # example script
   402        john echo "hello there"
   374      gstein exit 0
$
```

Details: revert

- Reverts changes made to a working copy
  - Replaces CVS's idiom of "**`rm file ; cvs update file`**"
- For safety, **`revert`** requires an explicit target and defaults to non-recursive operation

```
$ svn status
M      ./file2
M      ./moved-dir/file3
$ svn revert --recursive .
Reverted ./file2
Reverted ./moved-dir/file3
$
```

Details: info

- Provide information about files / directories

```
$ svn info file2
Path: file2
Name: file2
Url: http:// http://svn.example.com/repos/project/trunk/file2
Repository UUID: 65390229-12b7-0310-b90b-f21a5aa7ec8e
Revision: 16
Node Kind: file
Schedule: normal
Last Changed Author: gstein
Last Changed Rev: 13
Last Changed Date: 2004-06-16 07:34:53 -0700 (Wed, 16 Jun 2004)
Text Last Updated: 2004-06-20 08:58:20 -0700 (Sun, 20 Jun 2004)
Properties Last Updated: 2004-06-20 08:58:20 -0700 (Sun, 20 Jun 2004)
Checksum: 4fc8f533ca82f9f2b4137606f4668061
$
```

Details: properties

- Five different commands for manipulating properties on files and directories

```
$ svn propset test-property "hi there" file2
property 'test-property' set on 'file2'
$ svn proplist file2
Properties on 'file2':
  test-property
$ svn propget test-property file2
hi there
$ svn propedit test-property file2
```

editor pops up here

```
Set new value for property 'test-property' on 'file2'
$ svn propget test-property file2
this is the new property value set in the editor
$ svn propdel test-property file2
property 'test-property' deleted from 'file2'.
$
```

Details: merge

- Merges changes from two sources/revisions into a target
- Merging is a complex topic. However, we can definitely say Subversion makes the problem more approachable than CVS's merging via "**`cvs update`**"

```
$ svn merge –r 15:16 file2 file6
U      file6
$
```

Details: resolved

- Cleans up conflict files left from a conflict during "**`svn update`**" or "**`svn merge`**"

```
$ ls file6*
file6
file6.mine
file6.r15
file6.r16
$ svn resolved file6
Resolved conflicted state of 'file6'
$ ls file6*
file6
$
```

Note: Similar to CVS, Subversion inserts conflict markers into the conflicted source file ("**`file6`**" in this example).

Details: import

- Loads new content into a repository

```
$ svn import http://svn.example.com/repos/project/ \
    localdir trunk -m "initial import"
Adding         localdir/file10
Adding         localdir/file11
Transmitting file data ..
Committed revision 1.
$
```

Details: export

- Just like a checkout, but the **`.svn`** administrative subdirectories are omitted
- Keywords are expanded and newline translation is performed

```
$ svn export http://svn.example.com/repos/project/trunk
A  trunk/file11
A  trunk/file10
Checked out revision 1.
$ ls -aF trunk
./  ../  file10  file11
$
```

Details: switch

- Switch a working copy to a branch

```
$ svn info | grep Url:
Url: http://svn.example.com/repos/test/trunk
$ svn switch http://svn.example.com/repos/project/branches/issue-10
U  ./file2
Updated to revision 18.
$ svn info | grep Url:
Url: http://svn.example.com/repos/test/branches/issue-10
$
```

Details: cat

- Displays a particular revision of a file
- The “cat” name comes from the Unix tool for displaying files
- Two main modes of operation
  - Display an older version of a working copy file
  - Display a file directly from the server (no working copy)

```
$ svn cat -r341 http://svn.example.com/repos/hello.sh
#!/bin/sh
# example script
echo "hi"
$
```

Details: list

- Displays a listing of the files in a directory
- Typically used with a URL to explore a repository
  - WebDAV clients are also excellent tools for exploration

```
$ svn ls http://svn.example.com/repos/
README.txt
hello.sh
subdir/
$
```

Note: the **`--verbose`** (or **`-v`**) is commonly used for this command.

Server Administration

Repository Setup

- "`svnadmin create`" for the basic repository creation
- Edit your Apache configuration
  - Use mod_dav_svn
  - Standard Apache directives to set authentication and authorization
  - Subversion clients understands several HTTP authentication styles
- Set up hook scripts for the repository
  - Typical: send email for each commit or property change
- Set up regular maintenance scripts
  - Back up the repository
  - Clean out unused Berkeley DB log files
  - Rarely: clean out stale SVN transactions and WebDAV activities

Backing Up

- Subversion supports "hot backups"
  - No need to lock out commits
  - No need for downtime while backups are made
- Use "**`svnadmin hotcopy`**"
  - **`hot-backup.py`** is a helpful wrapper
  - After the copy is made, it can be moved off-system
- Some people have used incremental repository dumps
- Note that Subversion's repository is built on Berkeley DB
  - Enables the hot backups
  - Journaled, transacted storage system for safety

Choosing the Server

- Two choices: Apache-based or svnserve
- Primary difference is using SSL versus SSH for the security infrastructure
- Apache has a better integration story
  - Tools
  - Existing networks
  - Monitoring
- svnserve can fit in with existing SSH infrastructure