SUBVERSION
A version control tool
**OUTLINE**

- What is subversion?
- Basic commands
- help
- Import
- Checkout
- what is .svn directory?
- update
- other commands
- status
- diff
- log
- links
WHAT IS SUBVERSION?

- Subversion is a **centralized** system for sharing information.
- At its core is a repository, which is a **central store** of data.
- The repository stores information in the form of a **filesystem tree**—a typical hierarchy of files and directories.
- Any **number of clients** connect to the repository, and then read or write to these files.
- By writing data, a client makes the information available to others; by reading data, the client receives information from others.
Basic Commands

- Update your working copy
  - `svn update`
- Make changes
  - `svn add`
  - `svn delete`
  - `svn copy`
  - `svn move`
- Examine your changes
  - `svn status`
  - `svn diff`
- Possibly undo some changes
  - `svn revert`
- Resolve conflicts (merge others' changes)
  - `svn update`
  - `svn resolve`
- Commit your changes
  - `svn commit`
HELP COMMAND

- Here is the most important command you'll ever need when using Subversion: `svn help`.
- The Subversion command-line client is self-documenting—at any time, a quick `svn help subcommand` will describe the syntax.

```
$ svn help import
import: Commit an unversioned file or tree into the repository.
usage: import [PATH] URL
Recursively commit a copy of PATH to URL.
If PATH is omitted '.' is assumed.
Parent directories are created as necessary in the repository.
If PATH is a directory, the contents of the directory are added
directly under URL.
...```
**IMPORT**

- The *svn import* command is a quick way to copy an unversioned tree of files into a repository. *svn import* doesn't require a working copy, and your files are immediately committed to the repository. You typically use this when you have an existing tree of files that you want to begin tracking in your Subversion repository. For example:

```bash
$ svn import mytree \
svn://priloc.mscs.mu.edu/team1/myproj \
-m "Initial import"
Adding mytree/foo.c
Adding mytree/bar.c
Adding mytree/subdir
Adding mytree/subdir/quux.h
Committed revision 1.
```
The previous example copied the contents of directory `mytree` under the directory `myproj` in the repository:

```
$ svn list svn://priloc.mscs.mu.edu/team1/myproj
bar.c
foo.c
subdir/
```

- Note that after the import is finished, the original tree is **not converted into a working copy**. To start working, you still need to `svn checkout`.

- **Checkout** a fresh working copy of the tree.
CHECKOUT

Most of the time, you will start using a Subversion repository by doing a *checkout* of your project. Checking out a repository creates a “working copy” of it on your current working directory. This copy contains the HEAD (latest revision) of the Subversion repository that you specify on the command line:

```
$ svn checkout -username <userName> /svn://priloc.mscs.mu.edu/team1/myproj
```

A myproj/Makefile.in
A myproj/ac-helpers
A myproj/ac-helpers/install.sh
A myproj/ac-helpers/install-sh
A myproj/build.conf
...

Checked out revision 8810.
**WHAT'S WITH THE .SVN DIRECTORY?**

- Every directory in a working copy contains an administrative area—a subdirectory named .svn.
- Usually, directory listing commands won't show this subdirectory, but it is nevertheless an important directory.
- Whatever you do, **don't delete or change anything** in the administrative area! Subversion depends on it to manage your working copy.
- If you **accidentally remove** the .svn subdirectory, the easiest way to fix the problem is to remove the entire containing directory (a normal system deletion, not svn delete), then run **svn update** from a parent directory. The Subversion client will download the directory you've deleted, with a new .svn area as well.
UPDATE YOUR WORKING COPY

- When working on a project with a team, you'll want to update your working copy to receive any changes other developers on the project have made since your last update. Use `svn update` to bring your working copy into sync with the latest revision in the repository:
  ```
  $ svn update
  U foo.c
  U bar.c
  Updated to revision 2.
  ```

- In this case, it appears that someone **checked in** modifications to both foo.c and bar.c since the last time you updated, and Subversion has updated your working copy to include those changes.

- When the server sends changes to your working copy via `svn update`, a **letter code** is displayed next to each item to let you know what actions Subversion performed to bring your working copy up to date. To find out what these letters mean, run `svn help update`.
OTHER COMMANDS

- Here is an overview of the five Subversion subcommands that you'll use most often to make tree changes:

  - `$ svn add foo`
    Schedule file, directory, or symbolic link foo to be added to the repository. When you next commit, foo will become a child of its parent directory. Note that if foo is a directory, everything underneath foo will be scheduled for addition. If you want only to add foo itself, pass the `--depth empty` option.

  - `$ svn delete foo`
    Schedule file, directory, or symbolic link foo to be deleted from the repository. If foo is a file or link, it is immediately deleted from your working copy. If foo is a directory, it is not deleted, but Subversion schedules it for deletion. When you commit your changes, foo will be entirely removed from your working copy and the repository.
**Other Commands (contd.)**

- `$ svn copy foo bar`
  Create a new item bar as a duplicate of foo and automatically schedule bar for addition. When bar is added to the repository on the next commit, its copy history is recorded (as having originally come from foo). **svn copy does not create intermediate directories** unless you pass the --parents option.

- `$ svn move foo bar`
  This command is exactly the same as running `svn copy foo bar; svn delete foo`. That is, bar is scheduled for addition as a copy of foo, and foo is scheduled for removal. **svn move does not create intermediate directories** unless you pass the --parents option.

- `$ svn mkdir blort`
  This command is exactly the same as running `mkdir blort; svn add blort`. That is, a new directory named blort is created and scheduled for addition.
**STATUS**

- **svn status** prints six columns of characters, followed by several whitespace characters, followed by a file or directory name. The first column tells the status of a file or directory and/or its contents. The codes we listed are:
  - **A** item
    The file, directory, or symbolic link item has been scheduled for addition into the repository.
  - **C** item
    The file item is in a state of conflict. That is, changes received from the server during an update overlap with local changes that you have in your working copy (and weren't resolved during the update). You must resolve this conflict before committing your changes to the repository.
  - **D** item
    The file, directory, or symbolic link item has been scheduled for deletion from the repository.
  - **M** item
    The contents of the file item have been modified.
STATUS (CONT'D.)

- If you pass a specific path to `svn status`, you get information about that item alone:
  ```
  $ svn status stuff/fish.c
  D stuff/fish.c
  ```

- `svn status` also has a `--verbose (-v)` option, which will show you the status of every item in your working copy, even if it has not been changed:
  ```
  $ svn status -v
  M 44 23 sally README
  44 30 sally INSTALL
  M 44 20 harry bar.c
  44 18 ira stuff
  44 35 harry stuff/trout.c
  D 44 19 ira stuff/fish.c
  44 21 sally stuff/things
  A 0 ? ? stuff/things/bloo.h
  44 36 harry stuff/things/gloo.c
  ```
None of the prior invocations to `svn status` contact the repository—instead, they compare the metadata in the `.svn` directory with the working copy. Finally, there is the `--show-updates (-u)` option, which contacts the repository and adds information about things that are out of date:

```
$ svn status -u -v
M   *     44 23 sally README
M     44 20 harry bar.c
   *     44 35 harry stuff/trout.c
D     44 19 ira stuff/fish.c
A     0  ?  ? stuff/things/bloo.h
```

Notice the two asterisks: if you were to run `svn update` at this point, you would receive changes to `README` and `trout.c`.

This tells you some very useful information—you'll need to update and get the server changes on `README` before you commit, or the repository will reject your commit for being out of date.
SVN DIFF

- You can find out exactly how you've modified things by running `svn diff` with no arguments, which prints out file changes in unified diff format:

  ```
  $ svn diff
  Index: bar.c
  ================================================================================
  --- bar.c (revision 3)
  +++ bar.c (working copy)
  @@ -1,7 +1,12 @@
  +#include <sys/types.h>
  +#include <sys/stat.h>
  +#include <unistd.h>
  +
  +#include <stdio.h>
  int main(void) {
  - printf("Sixty-four slices of American Cheese...
  + printf("Sixty-five slices of American Cheese...
  return 0;
  }
  ```

- Removed lines are prefaced with -, and added lines are prefaced with +.
VIEWING LOG MESSAGES

$ svn log

---------------------------------------------------------------------
Added include lines and corrected # of cheese slices.
---------------------------------------------------------------------
r2 | harry | 2008-05-14 18:43:15 -0500 (Wed, 14 May 2008) | 1 line
Added main() methods.
---------------------------------------------------------------------
r1 | sally | 2008-05-10 19:50:31 -0500 (Sat, 10 May 2008) | 1 line
Initial import
---------------------------------------------------------------------

- Note that the log messages are printed in *reverse chronological order* by default. If you wish to see a different range of revisions in a particular order or just a single revision, pass the --revision (-r) option.
TortoiseSVN

- Download link: [http://tortoisesvn.net/downloads](http://tortoisesvn.net/downloads)
Links

Thank You