Version Control with Subversion

Introduction

- Wouldn't you like to have a time machine?
- Software developers already have one! — it is called version control
- Version control (aka *Revision Control System* or *Source Control System* or *Source Code Management*) is the art and science of managing information, which for software projects implies managing files and directories over time.
- A repository manages all your files and directories. The repository is much like an ordinary file server, except that it remembers every change ever made to your files and directories. This allows you to recover older versions of your data, or examine the history of how your files have changed.

Various open-source version control systems

- RCS (Revision Control System). Simple, text based system. Included in Linux and Unix systems by default. No remote access. No directory level access.
- CVS (Concurrent Versioning System). Built on top of RCS. Adds directory level access as well as remote access.
- Subversion. A modern CVS "replacement" that isn't built on top of RCS. Allows directory access, web access (via an Apache Web server module), remote access (via ssh or svn server). Uses a centralized model with mulitple access-control possibilities.
- Git. A distributed version control system. There is no central repository like in subversion. Everyone has a copy of the repository. More complex model to learn.
- Mercurial. Another popular distributed version control system.

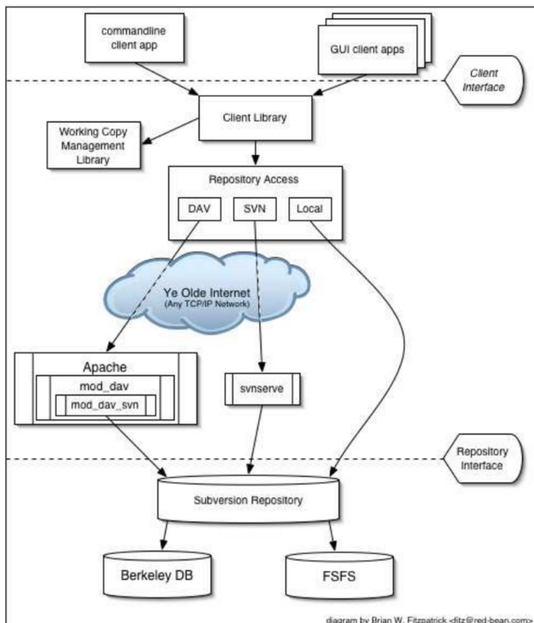
Comparison

- Subversion is the most widely used version control system.
- Subversion 66%, Git 33%, CVS 12%, Mercurial 10%.
  - Based on a survey of developers in the 2013 report from zeroturnaround.com.

Subversion Features

- Subversion has a client/server architecture. A developer interacts with a *client* program, which communicates with a *server* program, which accesses repositories.
- Multiple clients, communication protocols, repository-access mechanisms, and repository formats are available.
- Repository formats: Berkeley Database (deprecated in version 1.8) and FSFS (preferred and default).

Subversion Architecture



Versioning Models

The core mission of a version control system is to enable collaborative editing and sharing of data. But different systems use different strategies to achieve this.

- The Lock-Modify-Unlock Solution. The repository allows only one person to change a file at a time. To change a file, one must first obtain a lock. After you store the modified file back in the repository, then we unlock the file.
- The Copy-Modify-Merge Solution. Primary model in Subversion.
  - Each user's client contacts the project repository and creates a personal working copy–a local reflection of the repository's files and directories.
  - Users then work in parallel, modifying their private copies.
  - Finally, the private copies are merged together into a new, final version. The version control system often assists with the merging, but ultimately a human being is responsible for making it happen correctly.

Subversion also supports locking if it is needed. Typically locking is used for non-text files.

Downloading/Installing Subversion

We have subversion 1.8.x available in the lab on all workstations.

- On Fedora Linux, check the version with the command:
  `rpm -qa | grep subversion`
- If not using version 1.8 or newer, then get it using `yum` as follows:
  `yum -y update subversion* mod_dav_svn*`
- You can also download the source code from subversion.tigris.org and compile it directly.

Creating a Repository

- Creating a repository is done with the `svnadmin` command. This is an administrative task. A developer would not normally need to do this. We have created repositories for all students for the rest of your time in the CS program!
- For larger team projects that will have multiple supported versions, it is recommended that the top-level of the project have three subfolders by convention: `trunk`, `branches`, and `tags`. The subfolders aren't required but it is a common convention.
- For this class, you should be fine without having these subfolders.

```
mkdir project1
gvim project1/hello.c
```

Importing the Project into a Repository

- Now let's import the existing project named `project1` into svn.

```
svn import project1 \
 https://loginid@onyx.boisestate.edu/repos/students/loginid/project1 \
 -m "import the project"
```

- The `loginid` is your subversion user id with an associated password that was created for you by the administrator.
- The `-m "..."` option is a log message that is also stored and tracked by subversion. Log messages can be updated later (unless the admin disables that feature!) You can examine the log messages with the `svn log` command.
- We will abbreviate `onyx.boisestate.edu` to `onyx` in the rest of these notes. This will work on campus but from home you will have to use the full hostname for onyx.
- After importing a directory to the repository, you will need to check out a local copy. The original copy of your local directory will not be under version control. In fact, we can delete it since a copy is safely stored in the repository.
- Before we do that, let's first examine repository layout options and access mechanisms.

Checking-Out a Working Copy

This is a development task. A *working copy* is a private workspace in which you can make changes.

```
svn checkout \
  https://onyx/repos/students/loginid/project1/ \
  project1
```

Notes:

- The URL identifies the version to checkout.
- The last argument names the destination folder. If we skip it, svn makes it be the same name as the last component in the URL.
- The command `svn info` gives us detailed info about the current working copy. Try it!

Working on a Working Copy

You can now change your working copy. Let's change `hello.c` and add a **Makefile**.

```
cd project1
gvim hello.c # format nicely
gvim Makefile
svn add Makefile
svn status -u
svn diff
svn commit -m "make it nice"
```

Note that commit doesn't change the revision of the working copy. For that we need to do an update. We may also delay until later. Let's continue working on the project.

```
svn update
gvim hello.c # add stdio.h
svn status -u
svn diff
svn commit -m "add stdio.h"
```

Notes:

- The whole directory tree that we check-in gets a new version number.

Basic Work Cycle

- Get a working copy
  - svn checkout (co)
- Update your working copy
  - svn update (up)
- Make changes
  - edit files
  - svn add
  - svn mkdir
  - svn delete (rm)
  - svn copy (cp)
  - svn move (mv)
- Examine your changes
  - svn status (st)
  - svn diff
- Get information
  - svn info
  - svn log
- Commit your changes
  - svn commit (ci)
- Getting help on svn options.
  - svn help commit

Web Access to Repository

- We have the ViewVC software installed that gives us a web based browsing access to your svn repository.
- Try the following URL in your web browser:

  `https://onyx.boisestate.edu/viewvc/students/loginid`

  where `loginid` is your svn login id. It will ask for your svn password and then give you a nice web based interface to browse your repository.

Merges and Conflicts

Suppose two people are working on the same file.

```
cd project1
#add comment on top/bottom
gvim hello.c
svn commit -m "made some changes"
```

```
svn checkout https://onyx/repos/students/loginid/project1/ \
             project2
cd project2
gvim hello.c # add comment on top/bottom
svn commit -m "made some changes"
---fails due to conflict---
svn update
gvim hello.c  # fix conflict
svn resolved hello.c
svn commit -m "made some changes"
```

GUIs for Subversion

There are many GUI tools available for subversion. The following are recommended:

- **Subclipse Eclipse plugin**. First, check if you already have the Subclipse plugin under *Help* → *Software Updates* → *Manage Configuration*. Otherwise go to `http://subclipse.tigris.org/install.html` for step-by-step instructions on how to install the Subclipse plugin from Eclipse. The installation of Subclispe plugin requires that you have write access to the folder that contains the Eclipse installation. Subclipse is already installed in the lab.
- **TortoiseSVN**. is a nice stand-alone GUI for subversion that works on MS Windows. It integrates with the File Explorer.
- **KDESVN**. A feature rich client that integrates with the KDE desktop in Linux.

Subclipse Plugin for Eclipse

The subclipse plugin gives you most of the subversion functionality in Eclipse.

- We can use the SVN perspective to browse repositories. From there we can checkout a subversion project as an Eclipse project.
- Subclipse supports the `https://` and `svn+ssh://` protocols but does not support the `file://` protocol.
- A new menu *Team* gives you access to subversion commands from your project. All of the concepts we have covered can be accessed from the subclipse plugin except for the administrative commands.
- You can share an existing project in Eclipse using the menu items *Team* → *Share project...*. To share a project, you need to know the URL of an existing repository. This is the same as the *import* command we saw earlier.

References

- http://subversion.tigris.org/ Homepage for the subversion project.
- http://svnbook.red-bean.com Excellent book for subversion.
- Thanks to Jim Buffenbarger for providing me notes on subversion. These notes are based on his notes.