Source Control Management System (SCM) – git

Project management

- Difficult to keep track of different versions of a file if several people are making updates
  - Source code and documentation files change frequently
  - Bugs need to be fixed (but keep a copy of the older working program)
  - Programs may need to be enhanced
  - Software is *released* at non-regular intervals
- Problem is harder if there is more than one version of the program in circulation
- Different people may attempt to fix different bugs by modifying different parts in the same file
- Problems solved by utilities to manage and track changes in the files
  - Source code management systems – SCCS, RCS, CVS, SVN, git
  - Record a specific *version* or *revision* of the file so that it can be recovered in the event of a problem in that version
  - Usable for managing source code and software documentation
  - Control the set of people allowed to update a file
  - Record the name of the person who made the change, and a comment about what was changed (log commentary)
  - Provide control over new revision creation (major or minor revision), depending upon the version
  - Possible to regenerate any version of the file that was saved
  - Make sure that only one person can modify the file at any time

Goals of project control

- Ability to group files by revision (snapshot)
- Support for product abstraction
  - Structuring a group of source files
- Support for concurrent development
- Ability to rebuild product revisions
- Support for multiple target platforms
- Control of released files

git

- Created by Linus Torvalds as an open source project to maintain the code base for Linux
- Used to manage multiple revisions of files
  - Older versioning systems used *delta-based version control*
  - `git` uses *snapshots* to save versions
- Automates the storage, retrieval, logging, identification, and merging of revisions
- Saves old revisions in a space efficient manner

- Maintains a complete history of changes
- Resolves access conflicts
- Maintains a tree of revisions
- Controls releases and configurations

Creating personal git server

- Normally, you will be provided a git server to enable collaborative code development in an organization
  - When you are working on individual projects, you will need to create your personal git server
- Steps to create your personal git server
  - Set up a password-less ssh login

    ```
    ssh-keygen -t rsa
    ```

    * Use the default location to store the key
    * Specify a pass-phrase when prompted; remember the pass-phrase
    * Note down the location of the public key
  - If you are using a remote machine as git server, you need to copy the keys to the server (assuming username `ssoid` on remote server)

    ```
    cat ~/.ssh/id_rsa.pub | ssh ssoid@remote-server "mkdir -p ~/.ssh && \
        cat >> ~/.ssh/authorized_keys"
    cat ~/.ssh/id_rsa.pub | ssh bhatia@delmar.umsl.edu "mkdir -p ~/.ssh && \
        cat >> ~/.ssh/authorized_keys"
    ```
  - The following commands will get rid of passphrase query every time you log in remotely:

    ```
    exec ssh-agent bash
    ssh-add
    ```

    You will enter passphrase just once when prompted
  - `ssh` into the server as `ssoid` and create a project directory for git and create an empty repository

    ```
    mkdir -p ~/projectname.git
    cd ~/projectname.git
    git init --bare
    ```
  - `exit` to get back to your local machine
  - Create a git repository on local machine and initialize it

    ```
    mkdir -p ~/git/project
    cd ~/git/project
    git init
    ```
  - Add files into the repository

    ```
    git add .
    ```

    * Run the above command every time you create new files in the directory
  - Whenever you have a significant change in the file, you need to commit the change

    ```
    git commit -m "message" -a
    git commit -m "message" file.c
    git remote add origin ssoid@remote-server:project.git
        git remote add origin bhatias@delmar.umsl.edu:project.git
    git push origin master
    ```

- You can clone your remote project on a new machine

```
git clone ssoid@remote-server:project.git
    mkdir ~/tmp
    cd ~/tmp
    git clone bhatias@delmar.umsl.edu:project.git
```

Git workflow

- When setting up `git`, a new subdirectory with the name `.git` gets created
- Subdirectory contains three subdirectories
  - Working directory – directory that contains your files
  - Index – staging area for files that eventually get committed
  - Head – points to the last commit that was made.
  - The flow is your working directory, then add to staging area, then commit moves files to head.
- Git commands to move files through workflow
  - `cd project` Change directory to where your files are located.
  - `git init` Use this command to initialize your working directory the first time for use with git; After first time, you don't have to do this again.
  - `git add *` or `git add .` moves changes to index area
  - `git commit -m "commit message"` moves changes to head area; note that files are not in remote repository yet
  - `git push origin master` Use this command the first time
    - Compresses changes and uses `scp` to push files to remote repository
  - `git push` use this command after the first time
  - If you have a different remote server you want to push changes to use the following command

    ```
    git remote add origin loginid@remoteserver.com:project.git
    ```

Git Logs

- Git keeps logs of everything you do.
- Git commands for various log output.
  - `git log` Show latest changes
  - `git log --author=Chev` Show only changes by Chev
  - `git log --pretty=oneline` Shows compressed description of commits on one line.
  - `git log --graph --oneline --decorate --all` Outputs nice looking logs
  - `git log --name-status` Shows which files have changed with each commit
  - `git log --help` More options on output.

Git Tagging

- Used to name a particular commit

  - Useful for naming a public release of software.
  - `git tag 1.0.0 17948a9735`
    * `17948b9725` is 1st 10 digits of a commit id, which can be obtained by looking at the logs

Git and Recovering Deleted files

- Suppose you have deleted important work on your computer that you had already committed to `git`
  - Let's say this file is called `file.txt`
- This is the command to get your file back from remote server

```
git archive --remote=ssoid@delmar.umsl.edu:project.git HEAD file.txt | tar -x >  file.txt
```

- This is the command to get your file back form local cached copy

```
git checkout -- file.txt
```

- These series of commands will reset local cache and retrieve a files from remote.

```
git fetch origin
git reset --hard origin/master
```

Git useful commands and options

- `gitk` Graphical User Interface for git, linux systems
- `git config color.ui true` colorize output from git

Free Git Book

- `http://git-scm.com/book/en/v2`