Version Control Systems

Musadaq Mansoor

Version Control Systems (VCS)

- A version control system allows programmers to keep track of every revision of all source code files
  - The main element of the version control system is the repository, a database or directory which contains each of the files contained in the system
  - A programmer can pick a point at any time in the history of the project and see exactly what those files looked like at the time
  - Latest version of any file can be retrieved from the repository
  - Changing a file will not unexpectedly overwrite any previous changes to that file; any change can be rolled back

Why version control?

- For working by yourself:
  - Gives you a “time machine” for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- For working with others:
  - Greatly simplifies concurrent work, merging changes
- For getting an internship or job:
  - Any company with a clue uses some kind of version control
  - Companies without a clue are bad places to work

Problem of File Sharing

- All version control systems have to solve the same fundamental problem: how will the system allow users to share information, but prevent them from accidentally stepping on each other's feet?
- Scenario:
  - Two users read the same file from Repository
  - Both begin to edit their copies
  - User A publishes his work first
  - User B overwrites User A version

Two Common Models for VCS

- In a lock-modify-unlock system, only one person can work on any file at a time.
  - A programmer must check a file out of the repository before it can be modified. The system prevents anyone else from modifying any file until it is checked back in.
  - On large projects, the team can run into delays because one programmer is often stuck waiting for a file to be available.
  - Example Apache Subversion

Two Common Models for VCS

- In a copy-modify-merge system, multiple people can work on a single file at a time
  - When a programmer wants to update the repository with his changes, he retrieves all changes which have occurred to the checked out files and reconciles any of them which conflict with changes he made before updating the repository
  - Example: Git

Two Classes of VCS

- Distributed VCS
- Centralized VCS

Centralized VCS

- Centralized VCSs keep the history of changes on a central server from which everyone requests the latest version of the work and pushes the latest changes to
- Example: Perforce, SVN

Distributed VCS

- In a Distributed VCS, everyone has a local copy of the entire work’s history.
- This means that it is not necessary to be online to change revisions or add changes to the work.
- “Distributed” comes from the fact that there isn’t a central entity in charge of the work’s history, so that anyone can sync with any other team member.
- Distributed systems like Mercurial and Git are newer and are gradually replacing centralized systems like CVS and Subversion

VCS Common Terminologies

- **Repository**
  - A shared database with the complete revision history of all files under version control. It's a master storage to store multiple revisions of all the files
- **Revision/Version**
  - A committed change in the history of a file or set of files. This is the numerical reference supplied by the VCS to track the different editions it is holding of the file.

VCS Common Terminologies

- **Branches**
  - It’s a set of files which needs to be created at one particular time so that, from that time forward, two copies of those files may develop at different speeds or in different ways independently of each other. Branches are also known as “lines of development”
- **Conflict**
  - The situation when two members are trying to commit changes that affect the same region of the same file. These must be resolved either manually or by using a Merge tool

VCS Common Terminologies

- **Merge**
  - Combining multiple changes made to different working copies of the same files in the source repository. Merging is a strategy for managing conflicts by letting multiple developers work at the same time (with no locks on files), and then incorporating their work into one combined version

Why Git?

- Git has many advantages over earlier systems such as CVS and Subversion
  - More efficient, better workflow, etc.
  - Simple
  - Lot of documentation available

Download and install Git

- Link for Downloading: http://git-scm.com/downloads
- Here's one from StackExchange: http://stackoverflow.com/questions/315911/git-for-beginners-the-definitive-practical-guide#323764

- Note: Git is primarily a command-line tool but available in GUI based packages as well

Introduce yourself to Git

- Enter these lines (with appropriate changes):
  - `git config --global user.name "John Smith"`
  - `git config --global user.email` jsmith@seas.upenn.edu
- You only need to do this once
- If you want to use a different name/email address for a particular project, you can change it for just that project
  - `cd` to the project directory
  - Use the above commands, but leave out the `--global`

Create and fill a repository

1. `cd` to the project directory you want to use
2. Type in `git init`
   - This creates the repository (a directory named `.git`)
   - You seldom (if ever) need to look inside this directory
3. Type in `git add .`
   - The period at the end is part of this command!
     - Period means “this directory”
   - This adds all your current files to the repository
4. Type in `git commit -m "Initial commit"`
   - You can use a different commit message, if you like

Clone a repository from elsewhere

- `git clone` *`URL`*
- `git clone` *`URL mypath`*
  - These make an exact copy of the repository at the given URL
- `git clone git://github.com/`*`rest_of_path/file.git`*
  - Github is the most popular (free) public repository
- All repositories are equal
  - But you can treat some particular repository (such as one on Github) as the “master” directory
- Typically, each team member works in his/her own repository, and “merges” with other repositories as appropriate

`init` and the `.git` repository

- When you said `git init` in your project directory, or when you cloned an existing project, you created a repository
  - The repository is a subdirectory named `.git` containing various files
  - The dot indicates a “hidden” directory
  - You do *not* work directly with the contents of that directory; various git commands do that for you
  - You *do* need a basic understanding of what is in the repository

Making commits

- You do your work in your project directory, as usual
- If you create new files and/or folders, they are *not tracked* by Git unless you ask it to do so
  - `git add` *`newFile1 newFolder1 newFolder2 newFile2`*
- Committing makes a “snapshot” of everything being tracked into your repository
  - A message telling what you have done is required
  - `git commit -m “Uncrevulated the conundrum bar”`
  - `git commit`
    - This version opens an editor for you the enter the message
    - To finish, save and quit the editor

Commits and graphs

- A commit is when you tell git that a change (or addition) you have made is ready to be included in the project
- When you commit your change to git, it creates a commit object
  - A commit object represents the complete state of the project, including all the files in the project
  - The *very first* commit object has no "parents"
  - Usually, you take some commit object, make some changes, and create a new commit object; the original commit object is the parent of the new commit object
    - Hence, most commit objects have a single parent

Working with your own repository

- A head is a reference to a commit object
- The “current head” is called HEAD (all caps)
- Usually, you will take HEAD (the current commit object), make some changes to it, and commit the changes, creating a new current commit object
- You can also take any previous commit object, make changes to it, and commit those changes
  - This creates a branch in the graph of commit objects
- You can merge any previous commit objects
  - This joins branches in the commit graph

Commit messages

- In git, “Commits are cheap.” Do them often.
- When you commit, you must provide a one-line message stating what you have done
  - Terrible message: “Fixed a bunch of things”
  - Better message: “Corrected the calculation of median scores”
- Commit messages can be very helpful, to yourself as well as to your team members
- You can’t say much in one line, so commit often

Working with others

- All repositories are equal, but it is convenient to have one central repository in the cloud
- Here's what you normally do:
  - Download the current HEAD from the central repository
  - Make your changes
  - Commit your changes to your local repository
  - Check to make sure someone else on your team hasn't updated the central repository since you got it (Download current HEAD again)
  - Upload your changes to the central repository
- If the central repository *has* changed since you got it:
  - It is *your* responsibility to **merge your two versions**
    - This is a strong incentive to commit and upload often!
  - Git can often do this for you, if there aren't incompatible changes

Typical workflow

- `git pull` *`remote_repository`*
  - Get changes from a remote repository and merge them into your own repository
- `git status`
  - See what Git thinks is going on
  - Use this frequently!
- Work on your files (remember to `add` any new ones)
- `git commit -m` *“What I did”*
- `git push`

Keeping it simple

- If you:
  - Make sure you are current with the central repository
  - Make some improvements to your code
  - Update the central repository before anyone else does
- Then you don’t have to worry about resolving conflicts or working with multiple branches
  - All the complexity in git comes from dealing with these
- Therefore:
  - Make sure you are up-to-date before starting to work
  - Commit and update the central repository frequently
- If you need help: https://help.github.com/

Tutorial Link (Credits to Azhar)

- https://www.youtube.com/watch?v=3RjQznt-8kE&list=PL4cUxeGkcC9goXbgTDQ0n_4TBzOO0ocPR

Questions?

THE END