Version Control Workshop 1

Frederick Brunn
Project Manager

In this workshop we will cover...

- Basic version control concepts

- Getting started using version control in your digital project

Version Control

Why use version control?

- Keep track of who is doing what with the code
  - What features have we added?
  - What bugs are there / have been fixed?
- Be able to roll back any changes if necessary

Version Control Terminology

Repository

- The repository is where files' current and historical data are stored.
- Refers to both the metadata (history of changes and so on) and the files / previous versions of files being stored

Version Control Terminology

Commit

- A commit is an event wherein the most recent changes to a branch are written/saved.
- For the purposes of this tutorial, we do not need to discuss branches.

Version Control Terminology

Working Copy

- The working copy is the files in your repository/branch as they exist currently in your local folder.
- All file editing / creation / removal occurs within the working copy first.

Version Control Terminology

Checkout

- To checkout a commit is to load the repository as saved in that commit to be your working copy.
- Checkouts create a local working copy from the repository.

Version Control Terminology

Clone

- To clone a repository is to copy the repository and checkout the latest commit as your working copy.

Version Control Terminology

Pull

- To pull from a remote repository is to update your local repository with any additional commits that were pushed to the remote repository, and checkout the latest commit as your working copy.

Version Control Terminology

Push

- To push to a remote repository is to update the remote repository with any commits made on your local repository.

Version Control Systems

- Outside of development: Google docs / slides
  - Simple version control; no branching, one line of revisions
  - https://support.google.com/docs/answer/190843?hl=en

Version Control Systems

- Development
  - CVS
    - Concurrent Version System
    - Created in 1986
    - Defunct- no longer updated as of 2008, but sometimes encountered "in the wild"

Version Control Systems

- Development
  - SVN
    - Abbreviated SVN
    - Very strong community, but not used as often as Git

Why Git?

- It’s popular
  - Git - 60,975 questions tagged
    - http://stackoverflow.com/questions/tagged/git
  - SVN - 22,394 questions tagged
    - http://stackoverflow.com/questions/tagged/svn
  - CVS - 1,395 questions tagged
    - http://stackoverflow.com/questions/tagged/cvs
  - Mercurial - 6,811 questions tagged
    - http://stackoverflow.com/questions/tagged/mercurial

Why Git?

- It’s easy to use
  - GUI and non-GUI options for most platforms’
  - We will focus on using the console, but the same techniques can be used in GUI versions
  - Windows - TortoiseGit

Why Git?

- It’s portable
  - Git knowledge parallels version control knowledge

Git

- Created by Linus Torvalds in 2005
- Free, as in beer and as in speech

Git vs. Github / Bitbucket / etc.

- Git is NOT Github
  - Git is software for local version control
  - Github / Bitbucket are websites that offer remote hosting of git repositories

Create repository

- To create a repository, go to the folder of your project and initialize your repository.
- `git init`
- This creates a new folder ".git" that contains all of your repository files.

Clone repository

- To clone a remote repository, go to the folder where you want the project to be stored. Then clone the project using the https link
- `git clone <link>`
- Usually the clone link can be found on the repository page.

Commit to a repository

- To commit your current files, you first need to stage them for the next commit, then commit them to your repository.

Commit to a repository

- Staging
  - `git add <files>`
  - Git add stages the files listed as an argument. To stage all files and folders, use `git add .`

Commit to a repository

- Staging
  - You can stage specific files only using wildcards as well.
  - `git add *.c`

Commit to a repository

- Check that staging is correct
  - You can use git status to see the current files staged. You should do this before each commit to make sure that everything is in order.
  - `git status`

Commit to a repository

- Finally, commit to the repository
  - You only want to commit after you have added a new feature or fixed all current issues in your code, and your project runs perfectly fine.
  - `git commit`

Commit to a repository

- Finally, commit to the repository
  - You want to include a short message with each commit describing exactly what you changed. Use the -m flag to do this.
  - `git commit -m “Added new features and fixed bugs”`

Commit to a repository

- Finally, commit to the repository
  - You can also use `git commit --dry-run` in order to preview your commit, so if you have lots of flags or a more complicated commit statement you can make sure everything is as it should be without spamming commits.

Viewing a repository

- `git status` can also be used outside of making a commit to see what files have been changed.
- `git log` outputs a list of the commits on the repository in reverse chronological order. You can use `-2` to show only the last two entries.

Fixing Mistakes

- Forgot to stage some files before committing?
  - Make whatever changes you need and use `git commit --amend`
  - This preserves everything else in your commit such as messages

Fixing Mistakes

- Forgot to stage some files before committing?
  - `git commit -m "Added new file"`
    - Oops, we forgot to stage a file we changed
  - `git add forgotten.c`
  - `git commit --amend`

Fixing Mistakes

- Staged files that you don’t want to commit?
  - `git reset <file>`
  - Unstages the files to be committed

Fixing Mistakes

- Project files are FUBAR?
  - `git checkout -- <file>`
  - This restores your file given as it was during the last commit

Fixing Mistakes

- Project files are FUBAR?
  - `git checkout -- <file>`
  - This is a **DANGEROUS** command- it copies the older file over the newer one, so any changes you have made are gone

Remotes

- There is much more to be covered here, but for the purposes of this tutorial we shall cover the basics you need to work with just one branch.

View the current remotes

- `git remote`
- For most repositories this will just list "origin". You can add more using `git remote add <remote link>`
- Origin is the repository that you originally cloned the repository from.

Update repository metadata

- `git fetch <remote>`
- Pull the metadata (available commits elsewhere etc.) from the remote without making any changes to your working copy.

Update repository + working copy

- `git pull <remote> <branch>`
- Pulls the changes from that remote and merges them into your repository.
- The main branch is called master, so to pull the changes from your remote you use `git pull origin master`

Update remote with local changes

- After you have made commits, you can push them to your remote using `git push <remote>`
- `git push origin`

Update remote with local changes

- If someone else has pushed before you have a chance to, you must first pull and incorporate those changes into your repository with pull before you can push.

Next workshop

- Branches
- Comparing commits
- And more...