Version Control, and Project Organisation

Dan Goldsmith

May 2018

Introduction

Version Control.

- Version control is an awesome tool
  - Helps us keep track of your work
  - Provides a method for working with others
  - **Industry Standard**

Version Control

- *IMPORTANTLY: IT CAN SAVE YOU LOSING YOUR PROJECT*

What is Version Control

- System for managing changes to a collection of information
    - Documents
    - Websites
    - Code
    - *Lecture Materials*

Version Control Basics

- Call initial set of files version *1*
- After each set of edits we *Commit* the changes and create a new version
    - Version 2, 3 . . . Etc

Simple Version Control

- Consider:
  - You are Writing an individual project. It works but you have a new feature to add.
    - Copy the code into a new file (program_N+1.py)
    - Make the changes there.
    - If you break everything, then you can go back to original version

Simple Version Control

- For one person on one project it works but is inefficient.
    - If there are lots of changes you end up with lots of files
    - You have to *REMEMBER* which file is which.
    - Takes some management if there are many files.

Simple Version Control (Groups):

- What about Groups?
    - Person one makes some changes, Increase the version number
    - Person two gets the new file, makes changes, Increases the version number?

Simple Version Control (Groups):

- How do we deal with people working concurrently?
    - *Lock* the file so only one person can edit?
    - *Manual Merging*?

Version Control Systems (VCS):

- Provide software for managing all of this.
    - Automatic version numbering
    - (Mostly) automatic merging
    - Ability to review old versions etc.

Types of VCS (Centralised):

- Original Client-Server style:
  - One Master copy of the files (and history) on a central server
  - Files are downloaded, edited and changes submitted to the master
- CVS, Subversion etc.
- ISSUES:
  - Require Coordination, (How do we work offline)
  - Locking?

Types of VCS (Distributed):

- Each use has their own copy of the repository (there is usually also be a centralised repository)
    - Developers work on their own version of the repo
    - Changes are then *Merged* with other users / central repo
- Examples: Git, Mercurial, Bazaar

Distributed VCS

- As All work is done locally:
    - You can commit several changes without needing to talk to the server
    - You can VC all your steps, but only upload the master when finished
    - Without Locks, multiple people can work on the code at the same time (with some care)

Distributed VCS:

- Think of it like your Desk:
    - You have the file you are currently working on. (Working Directory)
    - You store this in a File as part of a Draft (Staging Area)
    - The Final version is stored in your draw. (Repository)

Practice What you preach:

- Almost ALL of my work is done in GIT.
  - Lectures, Labs tutorial
  - Coding
  - Personal Projects

GIT

GIT

- We will use GIT as our VCS
    - Written by Linus in ∼2005
    - Pretty much the industry standard
    - FOSS

GitHub

- Git is the Version Control system
- GitHub is a company that hosts repositories
    - There are alternatives:
        - Bitbucket
        - GitLab etc.

Git Workflow

1. Update from remote repository (pull)
2. Work . . . ..
3. Add files for next commit (add)
4. Commit local version (commit)
    - Goto 2?
5. Sync with master repo (git push)

Using Git.

- Two options
    - Command Line (do it)
    - GUI / Windows integration (Meh)

Git CLI: Creating a repository

```
$git init
```

Git CLI: Getting an Eexisting repository

```
$git clone <url>
```

For example

```
$git clone git@github.com:djgoldsmith/shape.git
```

Git CLI: Getting the status of the repository

```
dang@dang-laptop:~/Documents/Github/SHAPE/305$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   01-Github.md

Untracked files:
  (use "git add <file>..." to include in what will be comm

        #Javascript.md#
        .#Javascript.md
```

Adding files to the Commit

Tells Git which files should be stored

```
$git add <filenames>
```

Committing files:

- *Stores* the current state of the local repository
- We get a new revision number to work with.

```
$git commit -m "Message"
```

Keeping Control of Commits

- I like to keep each commit focused on a particular aspect
  - Separates the code and makes revision easier
- So Rather than 1 commit.
  - Fix typo, add new function, fix error in existing function
- Have 3 individual commits:;
  - Fix Typo
  - Fix Error
  - Add Function

Advnanced Features:

- Not covered, but you will find these useful concepts:
  - Merging
  - Branches
  - Forks

Your Turn

https://try.github.io/