Source Code Management

Aroon Chande

Lecture 3

January 15, 2019
Outline

• Source code management and version control

• Introduction to git

• Class GitHub
The problem...

Many people, working on (many) files in a codebase, at the same time, collaboratively
To solve this we need

A common platform is needed that everyone can access, collaborate and allows

- Roles to be assigned
- Changes to be tracked and combined
- Code conflicts to be dealt with
- Issues to be brought to everyone’s attention and discussed
- Provided ability to document everything
A codebase is not just code

• A “codebase” is all the files that go into a project

• In a normal project this includes:
  • Programs, scripts, and other code (Perl, Python, and R scripts for example)
  • Documentation
  • Build configuration and tools
  • Examples for new users
  • Test cases to ensure your code does what it is supposed to do after each edit
Does this sound familiar?

1. You are writing a script for a homework assignment and just need to add a few more things to finish

2. You make what seems like a small edit, but it breaks your script

3. You manually undo your changes...

4. ...But it is still broken!?
This semester’s problem

• Each student will generate a codebase throughout the semester but:
  
  • How do you keep track of it?
  
  • What is the latest and greatest version?
  
  • What if you accidentally delete all your work?
  
  • How do you share your codebase?
Potential solutions

- How do you keep track of it and what’s the latest and greatest version?
  - Just edit one big script that everyone works on?
- Copy and paste code snippets into a big notebook? You’re on the right track!
- Put it on Dropbox?
- The version that’s been copy and pasted and renamed with the newest date?
- The one you emailed yourself at 4AM the day it’s due?
Potential solutions

• What if you accidentally delete all your work?
  • Go through the 5 stages of grief, eat a pizza all by yourself, and don’t tell anyone?
  • Frantically email the TAs and get an extension?

• How do you share your work?
  • Email? – Doesn’t always work
  • Dropbox? – Blocked some places
Version control to the rescue

Use source code management tools, like `git`, for version control and file management
Version control

• A source code management (SCM) tool:

  • Keeps multiple version of everything in your codebase, all in one place

  • Requires a comment or description be given for every change made before you update the master copy

  • Can show you the differences between versions of a file

  • Allows everyone to edit anything, at the same time
History of version control systems

• 1972 – Source Code Control System (SCCS)
  • Human readable history files
  • File integrity checking
  • Delta files, or only the changed lines, for updates – saved lots of space!
  • Could only track text files, no binaries
  • Operations get slower for every new file or change being added

• 1982 – Revision Control System (RCS)
  • Based on the ideas from SCCS, scaled a little better

• Both systems could only edit one file at a time and were local only
History of version control systems

• 1986 – Concurrent Versions System (CVS)
  • Client-server model of data distribution, first popular (and F/OSS) SCM that supported networking
  • Multi-file editing per version with the concept of branching
  • Support for binary files
  • Multi-user, but not concurrently

• 2000 – Subversion (SVN)
  • Client-server model with diffs instead of files being transferred
  • Atomic operations(!)
  • Per file / folder versioning and binary support
History of version control systems

• 2005 – Mercurial (hg)
  • One of the first, widely used distributed version control systems
  • Designed to be highly scalable and performant
  • Multi-file, multi-user concurrent editing with branching
  • Used by Google and Facebook

• 2005 – Git
  • Created for use by the Linux kernel developers
  • The other widely used distributed version control system
  • Built for flexibility and scalability
Management styles

Everybody edits, where each collaborator can make changes directly to the shared master
Management styles

Managed integration, where the **integration manager** reviews and merges changes into the **master** repository.
Management styles

Corporate model, feature-specific groups with managers, and a leader in charge of final integration into the master repository.
Outline

• Source code management and version control

• Introduction to git

• Class Github
Introduction to git

• Git is a free and open source SCM tool (https://git-scm.com/)

• It was originally developed by Linus Torvalds, the creator of the Linux kernel, to manage the kernel codebase
  • Git is *opinionated*, just like Linus (https://www.reddit.com/r/linusrants/)

• The Linux kernel is a *massive* codebase
  >20 million lines of code
  ~37,000 files
  1000’s of contributors
Introduction to git

• A project in git is called repository, or a “repo”

• Files are added, or “checked in”, to a repo by committing your local changes to a shared master version

• A repo is copied by cloning or forking

• There can be multiple versions of a repo, all with different changes, called branches
Introduction to *git*

- The integrity of the files in your repo is guaranteed
  - You will get exactly the same file out of git as you put into git

- Git is a distributed version control system
  - You make changes to a local copy and share your changes
  - *An internet connection is not required to work*
Introduction to git

• Git repos are sometimes viewed as trees

• There’s a **master** branch that is like the trunk of the tree, the source of each subsequent branch

• Branches can be split from the master branch or other branches

• We use branches for working on and staging fixes and new features
Introduction to **git**

![Diagram showing git branches]

- **Master branch**
- **Branch with bug fix**
- **New feature branch**
Introduction to git

• When you **clone** a repo, you download the version and history of every file from the server
  • Changes you make get sent back to the same server/repo

• When you **fork** a repo, you make a new copy the current version and history of every file from the server **and** claim ownership of the repo
  • Changes you make get sent to a different master copy on the server
  • Your copy no longer has the same changes as the original repo you forked
Introduction to git

• The work done on two branches can be merged together

• This process saves the history of each branch, then tries to combine changes

• Changes to the same file in both branches may result in conflicts
Introduction to *git*

Git follows the Work-Stage-Commit lifestyle

- *Work* on files in a repo you’ve cloned or forked
- *Stage* your work (*git add*)
- *Commit* describe your changes, saving a snapshot in time (*git commit*)
Introduction to git

• **You push** your local changes (commits) to the server
  • This allows other contributors to view your work

• You can **pull** other users’ changes from the server copy into your local copy
  • This is like merging two branches but instead merges forward and backward in history
Introduction to git
Introduction to git

• A special case of pulling, `git fetch`

• `git fetch` downloads and informs you of any changes but does not apply any changes, if present

• Fetch lets you answer the question, “Has anything changed on the server?”, without fear
Introduction to git: a simple repo

→ week-1 on ~/master ls -Al
total 20
  drwxr-xr-x 8 achande3 jordan 4096 Aug 22 14:46 .git
  -rw-r--r-- 1 achande3 jordan 230 Aug 21 07:35 .gitlab-ci.yml
  -rw-r--r-- 1 achande3 jordan 8479 Aug 21 07:33 README.md
  drwxr-xr-x 2 achande3 jordan 108 Aug 21 07:41 submission_testing

→ week-1 on ~/master  🎯
Introduction to git: a simple repo

```bash
$ grading-ci$ git branch -a
* grading-ci
  master
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
$ grading-ci$
Introduction to git: A bigger repository

The Bioconda git repository, a collaborative science software distribution system with 489 users and >1000 packages
Introduction to git

• You will be using git from the command line and the Github web UI (https://github.gatech.edu/compgenomics2019) for this class

• You can edit files from the website.....but don’t
  • Save this mostly for text files (READMEs)
  • Editing scripts on the website frequently breaks things
  • This semester your grade will be affected if you edit code on the web interface
Introduction to **git**: A few tips

- Commit often, push infrequently
  - For each meaningful change save (commit) your history, push at end of session

- Write good commit messages
  - “Fix”, “Update”, “edits” are all bad commit messages
  - Be short but descriptive. “Fixes issue #35; >1 period in file name breaks script”
Bad commit messages
Better commit messages

Commits on Sep 13, 2017
- Fix some help text typos
  - adich committed on Sep 13, 2017
- Remove UTF8 characters to bring py2 compat without extra envs and pun...
  - adich committed on Sep 13, 2017
  - jnew version to py3: closes #24

Commits on May 21, 2017
- Insert mode ate my quotation mark
  - adich committed on May 21, 2017
- After some nulling, we’ll go with textmode for gz handling in Py3
  - adich committed on May 21, 2017
- Simplify to always use text mode in py3 #32
  - adich committed on May 20, 2017
- Move ldz check into the correct conditional
  - adich committed on May 20, 2017
- Catch Ubuntu 14.04 + Python3 to address #32
  - adich committed on May 20, 2017

Commits on May 17, 2017
- Better handling of errors while building from pubMLST (#31)
  - adich committed on May 17, 2017
  - Colored stdout!
  - Mostly addresses #30

https://github.com/jordanlab/stringMLST
Introduction to git: A few tips

• Test before you push
  • Try not to push broken code, especially to master

• Learn how to reset your branches and don’t be afraid to restart
Outline

• Source code management and version control

• Introduction to git

• Class Github
Class Github

https://github.gatech.edu/compgenomics2019

• This site will contain all your code and other group work

• This data will remain private during class and will be open sourced at the end of the semester
Github assignment

• Visit Setting up your account repo
  • https://github.gatech.edu/compgenomics2019/people

• Create a new fork

• Let us know about yourself

• Stage, commit and push your work. Then create a merge request
Github assignment

• Once I’ve reviewed your merge request, I’ll either:
  • Accept it, merge your changes
  • Or ask you to make some changes

• Feel free to explore the features available to you
Questions?