Basics of version control

Matthew Evans

Part II Computational Physics, Lent 2019

Contents

Introduction

What is version control and why should I care?

Put simply, a version control system (VCS) (a.k.a. revision or source control) is a system that tracks and manages changes to a set of data (e.g. source code). The Emacs website summarises VCS by three capabilities:

- Reversibility: the ability to back up to a previous state if you discover that some modification you did was a mistake or a bad idea.
- Concurrency: the ability to have many people modifying the same collection of files knowing that conflicting modifications can be detected and resolved.

- History: the ability to attach historical data to your data, such as explanatory comments about the intention behind each change. Even for a programmer working solo, change histories are an important aid to memory; for a multi-person project, they are a vitally important form of communication among developers.

Say you are writing a report; instead of renaming different versions of files `report.tex`, `report_final.tex`, `report_FINAL.tex`, `report_FINALFINAL.tex`, `report_FINAL_abcdef.tex`, ad infinitum, version control systems allow you to save the entire history of a file as a series of staged changes, often called commits or revisions. This may not seem so useful for a linear process such as a single person writing a document, but for the non-linear process of software development, version control is a must. Every single serious software project exists under some form of version control, almost by definition. Learning effective version control techniques is a vital skill, not just careers in the tech industry, but also for producing reproducible code and thus reproducible science.

The aim of this tutorial is to teach basic concepts of version control that you might consider using for the coursework. A few hours of investment, and maybe some moments of confusion, will hopefully lead to a productivity boost. We shall work with the Git (`git`) version control system as, at the time of writing this, it dominates the market, with Subversion (`svn`) and Mercurial (`hg`) lagging behind, as evidenced by e.g. Google trends.

What is Git?

Git was created by Linus Torvalds in 2005 [1] to manage the source code of the Linux kernel, with the requirements that it be fast, distributed and secure. At the time, the Linux kernel was 6 million lines long [2], with thousands of developers worldwide [3]; whilst famous for creating Linux, many argue that Git is Linus Torvalds' greatest technical achievement.

`git`, like its forebears `svn` and `hg`, is what is called a distributed version control system. This means that there is no "master copy" of a project, and instead the entire history of a project is mirrored on the computer of every developer (and potentially user). This becomes extremely useful when multiple people are actively developing a project for reasons we shall touch on later.

Git has somewhat of a reputation for being difficult to learn and master, due its 21 different subcommands, and their myriad options. Most simple use cases, however, require only a few of these commands. A complete and in-depth documentation of their operation can be found at the command-line by running `git help <subcommand>`, whilst a brief summary on the subsection of Git that we will discuss can be found in

[1] Linus Torvalds started writing Git on April 3rd, it was then hosting its own source code by April 7th (source), and then was hosting the entire Linux kernel (2.6.12-rc2) by April 16th.

[2] According to the spike in 2005 on the Linux kernel's code frequency graph on Github, which is actually so large it breaks the rendering of the scale (each division is 1 million lines of code).

[3] ~14,500 people posted on the Linux kernel mailing list between 1995 and 2000, according to this report.

the cheatsheet in the appendix. Several graphical user interfaces (GUIs) also exist for Git, which you may prefer, though I do not have one to recommend.

What is a repository?

A repository (or repo) is a top-level directory of files and directories that is managed by a version control system, containing e.g. the source of an entire software project. Often, the term repository refers to a location on a remote server that maintains a central copy of the project that developers can push changes in their *local repository* to, and pull other people's changes from. All of the project history is stored in the `.git` folder at the top-level of the repository.

In the past, developers would often directly push to or pull from each other's local repositories, but nowadays most projects will host their VCS on either a private server, or trust of the many web-based version control service providers. The main players in this field are GitHub, BitBucket and GitLab, which each provide similar web interfaces to Git (or otherwise) repositories. These web services are very useful to individual users who don't want to run their own server, and in fact this very tutorial is hosted on GitHub. Open source software has really benefited from these web-facing services that provide centralised, discoverable web pages for projects, as well as ways to talk to and raise issues with developers, lowering the bar to making contributions yourself. Additionally, web-based services allow for the automation of many useful checks on software, as we shall see in the test-driven development section.

Each provider has its own advantages and disadvntages to consider but for our usage they are all broadly similar (and all allow unlimited private repos for free, often providing extra benefits for students [4]). One of the useful features of distributed VCS is that you can easily transfer your entire code history to a different provider, since you are constantly mirroring your own version of the project locally [5].

What is a commit?

- Hashing - first hash collision
- descriptive tag of incremental change.

[4] For example, GitHub offer the free student developer pack which provides a lot of resources on third-party plugins and graphical interfaces to Git, and BitBucket provide their own BitBucket education accounts.

[5] When Microsoft acquired GitHub in late 2018, giving them soft power over a large majority of open source software, anyone who objected to Microsoft's business practices could simply point their repository at a new remote to move all commit history.

Basic usage: single-user repositories

Setup

A simple example

Remote version control

More advanced usage

Branching, merging and collaboration

Test-driven development

Appendices

Basic subcommand cheatsheet

`clone`

Make your own local copy of a repository.

```
git clone https://github.com/ml-evs/part2-computing-git-tutorial.
```

`init`

Make a new empty repository in the current directory (i.e. sets up `.git` folder).

```
git init
```

`status`

Tell me which (if any) tracked files have been changed in my current working directory, since the last commit.

```
git status
```

`add`

Register the current state of this file with Git, without comitting it (yet).

```
git add version_control.md
```

`commit`

Create a hash for the existing changes, and tag that hash with a useful "commit message". If called without a filename as an argument, the commit will include all changes that have been *staged* (call `git status` to check this). The flag `-a` will commit *all* changes to tracked files in the repo. This command will open your editor to write the message (determined by environment variable `EDITOR`); alternatively, the message can be provided at the command line with the `-m` flag (see below).

```
git commit version_control.md -m 'Added "commit" section to the subcommand
 cheatsheet'
```

`diff`

Show the uncommited (well, strictly unstaged) changes to all files in the repository, or for a particular file if requested, as a "diff" (i.e. the difference) between the current state and previously commited state (by default, can alternatively view changes between any two previous commits).

```
git diff version_control.md
```

`push`

Uploads the history of the local repository to a pre-configured remote server (see example 1), after first verifying that there are no clashes (i.e. that the server does not possess extra commits to the same branch as being pushed).

```
git push
```

`pull`

Download any "new history" on the remote server for this repository, making sure that there are no clashes with local changes.

```
git pull
```

`checkout`

If called on a file that has been modified, delete those changes and rever the state of the file back to the last commit (the `HEAD` state). Also used when branching (see `git help checkout` for more).

```
git checkout version_control.md
```

`log`

Prints the list of historical commit messages and their hashes, either those that applied to a particular file, or if called without argument, the entire repository.

```
git log version_control.md
```

Useful links

- The Git website has lots of useful resources for learning Git in more detail.
- The Git source code is itself hosted on Github.
- Atlassian (owners of BitBucket) provide a more thorough cheatsheet of git commands
- GitHub student developer pack, BitBucket Education for free stuff.
- The Sustainble Software Institute is a UK-wide push for improving the quality of research software, along with the Research Software Engineer (RSE) movement to create new jobs titles for those in academia working predominantly on software.

Footnotes