Git: A Guide for Economists

Frank Pinter

22 February 2019

Outline

What is version control?

Version control is a way to keep track of changes to code, text, and documents. And data and outputs.

- It gives you an organized revision history
- It lets you experiment without fear
- It lets you go back and forth between many different versions of the same file, and see a list of the differences
- It makes (the technical aspects of) collaboration a breeze
- It lets you and your collaborators work on different versions and then merge them

What is Git?

- Git is a program that does version control
- It is the most popular version control program in software development
- It is easy to set up and get started
- There are many programs that add intuitive interfaces on top of Git
- Git integrates seamlessly with online collaboration tools like GitHub and GitLab

Table of Contents

Life without version control

Do you keep every variant of every program you ever wrote on a project?

- The code and the outputs?
- What if you discover a coding error? Which versions are right?
- How do you organize all the files?

Or, worse, do you only keep the latest thing you tried?

- What if you introduce a new mistake?
- What if you're experimenting and you accidentally keep the experiment in?

Version control 0.1: putting dates on things

Does this look familiar?

`run_regs_11_17_2018_v4_final_final.do`

"Not one piece of commercial software you have on your PC, your phone, your tablet, your car, or any other modern computing device was written with the 'date and initial' method." (Gentzkow and Shapiro)

Version control 0.2: Dropbox

- Dropbox keeps a crude version history.
  - But there are no labels or comments, and it's not easy to see the differences between files.
  - So if you want to dig up "the version where I had that other variable" you have to manually look through a bunch of versions.
  - And good luck if you changed two scripts, not just one.
- Dropbox lets you and your collaborators stay in sync.
  - What if you and your coauthor try to change the same script at the same time?
  - What if you are trying one change and, at the same time, your coauthor is trying a different change?

Version control 0.2: Dropbox

A Post It note spotted on a grad student's desk:

> *Don't forget! At 10:18 am on November 17th, we changed the specification to add new variable.*

Don't live this way.

Table of Contents

Why use Git?

Git is the dominant version control system today. There are others, but they're generally more work with no benefit.

Getting started with Git

1. Install Git (Linux, Mac, Windows)
2. Git comes with a command line interface (powerful!). You might want to add a graphical interface to make things easier:
   - GitKraken
     - The examples in this presentation use GitKraken
   - GitHub Desktop
   - RStudio (for R projects)

Getting started with your project

If you're starting your own project in GitKraken:

- Open GitKraken and select "Start a local project" or "Start a hosted project" (will get into this later)
- Choose the name and the local directory to use
- Start working in the directory

If your coauthor started a project, added you to it online, and you're putting it on your own computer:

- Open GitKraken and go to File $\rightarrow$Clone
- Select the service (GitHub/GitLab/Bitbucket), log in if necessary, and select the project from the list
- Choose the local directory to save it to

What actually is the Git repository?

- The Git local repository is associated with a particular directory
- Open the directory in your Git interface to see your options
- Git stores all its workings in that directory in a hidden subfolder called ".git"
- Corollary: don't use Git inside a Dropbox shared folder!
- Sharing is what remote repositories and hosting services are for (more on this later)

What should I include?

1. At a minimum:
   - Code (`.do`, `.R`, `.m`, `.jl`, and so on)
   - Text files (`.txt`)
   - LaTeX documents (`.tex`)
2. I also recommend:
   - Raw `.csv` datasets, if small (<10 MB)
3. These are binary files, so you can't see differences between versions. I recommend including them anyway.
   - PDF files
   - Word, Excel, PowerPoint files
4. Some people also include all datasets.
   - Note that GitHub doesn't allow files larger than 100 MB, or projects with total size larger than 1 GB.

For datasets, look into Git Large File Storage.

What should I exclude?

In order to avoid driving your coauthors crazy, you **must** tell Git to ignore the junk files using a file called *.gitignore*. It looks like this:

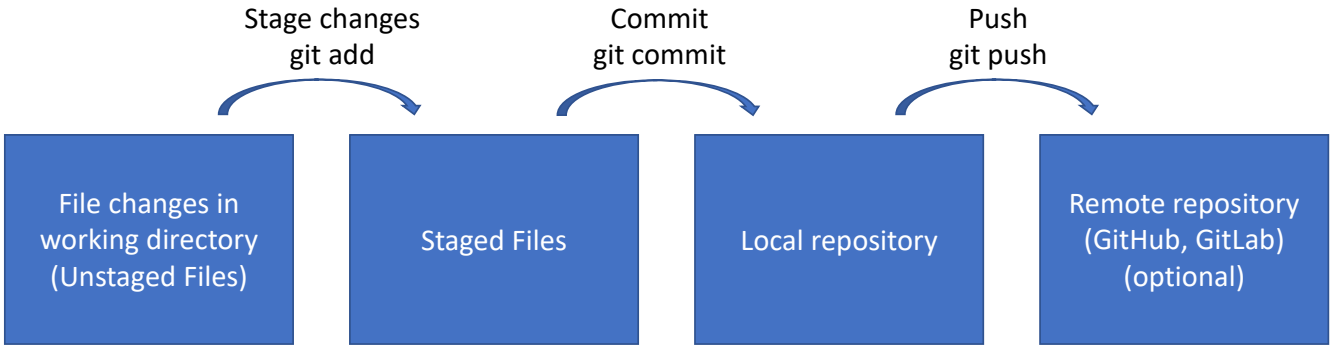
```
# Junk created by LaTeX
*.synctex.gz
*.out
*.log
# Junk created by R
.RData
# Junk created by Python
*.pyc
```

Best practice: use *.gitignore* to explicitly exclude *everything* that you don't want to include, and commit *.gitignore* like any other regular file.
GitHub maintains a list of standard *.gitignore* files for many common languages.

Table of Contents

The Git model

1. You do work in your **working directory**.
2. Then you add it to your **staging area**.
3. Once you've staged all your changes for one discrete task, **commit** a snapshot of the staging area.
4. If you have a remote repository, **push** your commit to the remote repository.



Commits: saving a snapshot

What is "one discrete task"? A collection of changes, across multiple files, that does *one thing*. Examples:

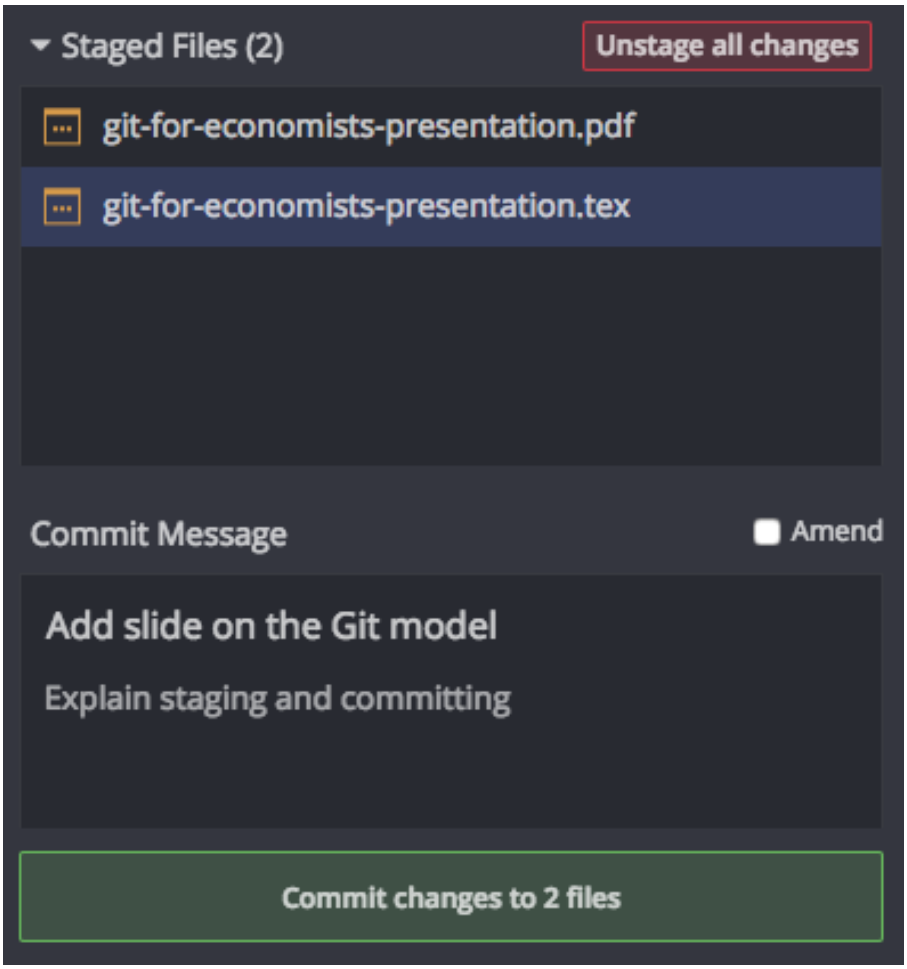
- Change the formatting of a variable from string to numeric, and treat it properly across multiple scripts
- Change your regression specification in code, in the output, and in your paper and supporting documentation
- Add a new function, and tests for that function

These form one **commit**, which you annotate with a detailed **commit message**. Examples:

- "Change the formatting of start date variable from string to Stata date format"
- "Add year dummies to regression specification"

The more detail, the more your future self will thank you.

Commits


Staged Files (2)
Unstage all changes
git-for-economists-presentation.pdf
git-for-economists-presentation.tex
Commit Message
Amend
Add slide on the Git model
Explain staging and committing
Commit changes to 2 files

Run tests before you commit

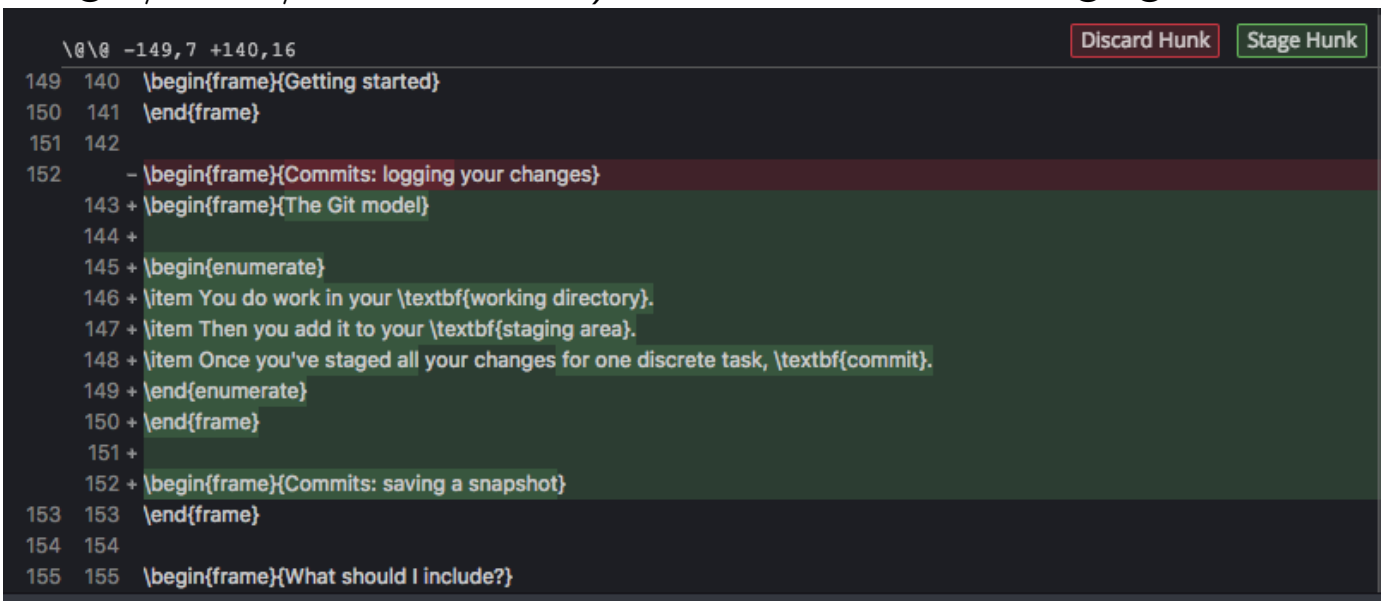
Your code should run properly when you commit.

- No runtime errors
  - Test this by running all code that changed, and everything that depends on it
  - Makefiles automate this process
  - Only skip if you are sure you didn't change anything important
- No compilation errors (including LaTeX)
- All your tests should pass
- Output should be consistent with what you've written
  - Don't report a negative regression coefficient, and write in words that the estimated coefficient is positive

But it's better to have frequent commits (that might have small mistakes) than to have giant, infrequent commits.

Viewing changes when committing

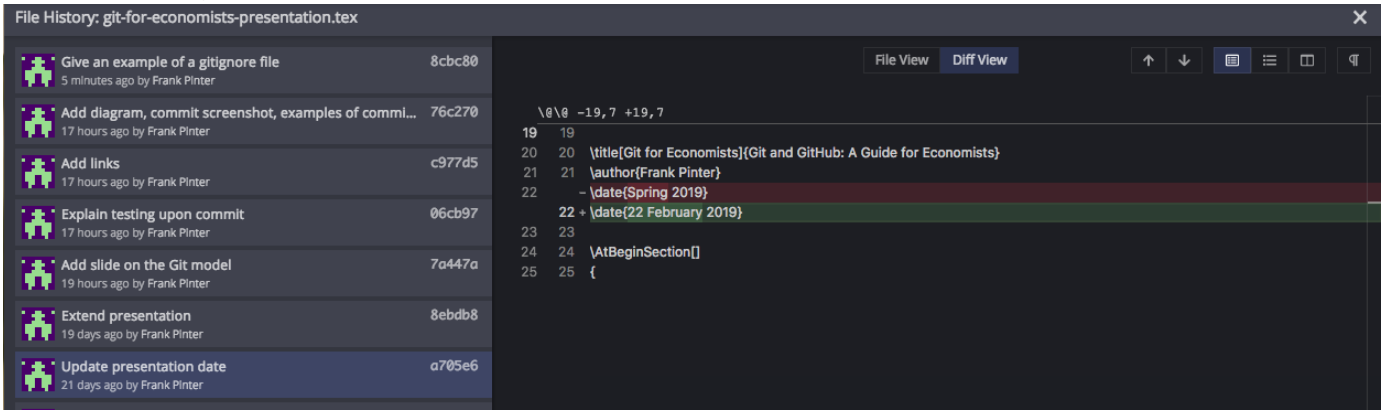
Git easily lets you see what changed in LaTeX, code (not images/PDFs/most datasets). Review this when staging!



(This is the GitKraken interface, but it looks similar in any other interface)

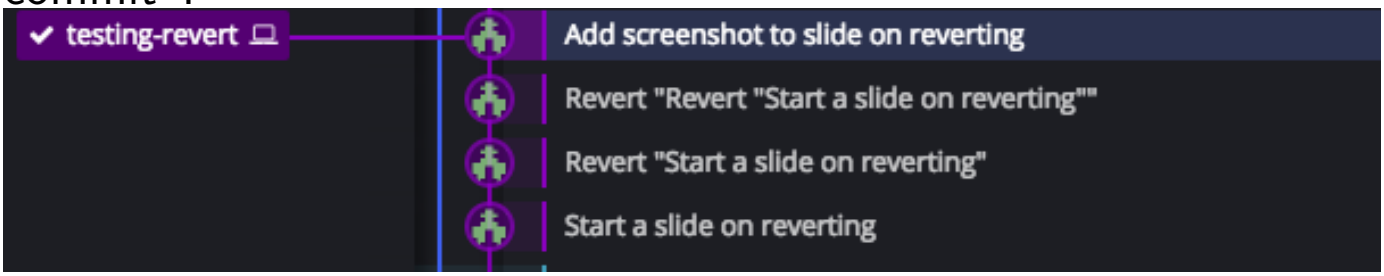
Viewing history

GitKraken shows you a list of past changes. You can also see just the history of a particular file:



Undoing history

What happens when a commit was a mistake? Revert it, to make a new commit that undoes it.
In GitKraken, right click on the commit line and select “Revert commit”.



Branches: trying things out

Branches are the most powerful part of Git

- By default, all the work you do goes into the "master" branch
- Want to experiment? Start a new branch
- You can switch between branches, and make commits to either branch
  - There is a catch: if you don't include your intermediate/final datasets in Git, you may need to re-make them when you switch
- If your experiment works out, commit and merge back into the master branch
  - If there are conflicts between the commits you've made on the two branches, Git will ask you to resolve them
  - This is easiest with a graphical interface like GitKraken
  - This can't be done with binary files (PDF, images, Word, Excel). You'll just have to decide which one to keep.
- If your experiment doesn't work out, delete the new branch painlessly

Keeping it local vs. using a remote repository

Git doesn't require a remote repository. You can run it 100% on your computer, with no connection to an outside server.

- This is useful if you have restrictions on your code (for example, you work with confidential health data)
  - Ask me if you have questions about using Git this way on the NBER cluster
- But a remote repository helps you keep things backed up seamlessly, and lets you collaborate with others
- You can push all your branches to the remote repository, or only some of them

Table of Contents

Remote repository

The remote repository is on a server, and holds a record of your commits and branches

- You **push** to the remote repository to save all your commits
- You **pull** from the remote repository to load all new commits
- Always commit before pushing or pulling
- If what you're doing is an experiment, make a new branch to avoid any trouble for your coauthor
- As with branches, if there are conflicts between your commits and your coauthor's commits, Git will ask you to resolve them

Hosting services

To collaborate, you'll need a service to host your remote repository.

- Here are a few:
  - GitHub (most popular)
  - GitLab
  - Bitbucket
- You can choose *public* (published for the world to see) or *private* (best for research in progress)
- Most services have restrictions on private repositories
- It's easy to use one service for one project, and another service for another project
- All these services have nice web interfaces for managing your project
- Some also have integrated task management systems

Learning the command line

- There are many more features that are best accessed from the Git command line
- And in some situations (like the NBER servers) you don't have a choice
- A fantastic resource for learning the command line: Jesús Fernández-Villaverde's notes on Git (`https://www.sas.upenn.edu/~jesusfv/Chapter_HPC_5_Git.pdf`)
- See also his class!

Conclusion

- At its simplest, Git is a way to keep track of the history of your work, and easily go back to past versions
- But it can be so much more!
- Experiment without fear
- Collaborate with far less back-and-forth
- The best way to learn Git: use it!

Further reading

- Again, Jesús Fernández-Villaverde's notes on Git (`https://www.sas.upenn.edu/~jesusfv/Chapter_HPC_5_Git.pdf`)
- Hadley Wickham's book on writing R packages. The chapter on Git and GitHub (`http://r-pkgs.had.co.nz/git.html`) is well-written and not specific to R.
- If you want to drill down on workflow, see the tutorial "Understanding the GitHub flow" (`https://guides.github.com/introduction/flow/`)