Version control, `Git` and GitHub

Monday, September 16

Content

1. The benefits of version control

Version control is an essential concept in collaborative software engineering projects. It is a set of tools that allows you to keep track of changes in your code, your data, and your files when you and your collaborators are working in parallel.

The issue with standard cloud-based storage platforms

Many of us are familiar with Dropbox, Google Drive and other cloud-based storage platforms. We use them to store our Stata/R scripts, datasets, slides and drafts when working on a project with our collaborators. Version control architectures, like `git` and version control platforms like GitHub, allow us to store information and collaborate, plus controlling the evolution of a project very tightly.

To see how using cloud-based storage can become awkward, consider the Dropbox folder of a typical social science research project. It usually ends up containing files with names like:

```
draft.docx

draft_16092019_JY.docx

draft_newResultTable_16092019_JY_.docx

draft_17092019_AD_.docx
```

Because multiple collaborators edit files independently and start working from different files, they lose track of the changes (For example, did AD use `draft.docx`, `draft_16092019_JY.docx` or `draft_newResultTable_16092019_JY_.docx` when he started working his version of the document?). They also need to send multiple emails to keep their collaborators updated about their contributions. When we use this way of organising files, we do not really know when to edit an old file and save it under its original name, or create a spin-off version of it, under a new name.

This problem gets even more serious when dealing with datasets and scripts for data analysis. The social scientist's workflow requires them to clean data, merge datasets, use several econometric specifications, and generate a multiplicity of outputs. The interdependencies between these activities mean that every mistake in early steps has knock-on effects on later ones.

Imagine your co-author and you have both cleaned the data in slightly different ways: you used `preProcessData.do` and she used `preProcessData_noOutliers_17092019_JB.do`. Unfortunately, the clean datasets output by these .do files are saved under the same name: `dataFinal.dta`. It is then impossible to know which version of the data is used by your analysis .do file: `IVreg.do`, for instance.

Moreover, your collaborator may have dated her .do file with `_17092019`, but the system says it has been edited several days later. It can be confusing.

This workflow is painful, and makes it harder to reproduce your own results. Version control tools are a solution to this.

It enables us to:

- Have one authoritative version of the project folder, at all times
- Work at the same time on the **same** piece of code, and reconcile discrepancies later
- Easily revert changes made by your collaborators or yourself
- Make clear who has changed what

- Manage the project efficiently (with functionalities such as flagging issues, allocating tasks, and communicating through the version control platform)
- Easily reproduce scientific results

2. The version control workflow

Doing version control ('VC' hereafter) properly requires a few changes to how we do things with cloud storage. The general process is simple:

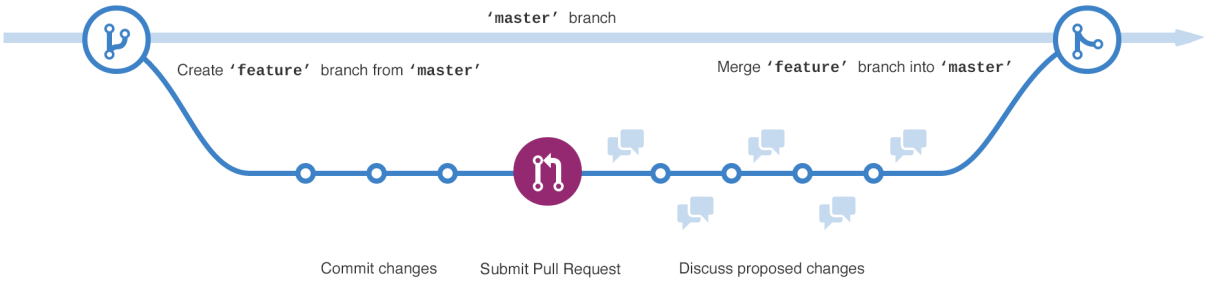
- (1) Download a copy of the main folder on your local machine
- (2) Do your work locally
- (3) When you are done, upload your local copy of the folder
- (4) Resolve any conflicts between your version and your collaborators' versions

We go into more details below.

Note: Programmers do not necessarily need to be online to do version control. So we are a bit oversimplifying when we say that you 'download' or 'upload' a folder. Version control is simply a method to collaborate efficiently, or even keep track of modifications in a single-user project. This can be done offline, and we use a more precise language below, when we explain the VC workflow. However, as most research projects are managed online, and as this makes the analogy with Dropbox easier, there is no real harm in using 'download' or 'upload' at this stage.

2.1. VC Lingo

The graphic below shows how one makes changes in the shared folder.



Graph from [GitHub Guides (https://guides.github.com/introduction/flow/)](https://guides.github.com/introduction/flow/)

(1) First of all, one of the collaborators sets up a folder where all the files will be stored. This folder is called a **repository** (or 'repo'). When downloaded to someone's machine, the repository is a just normal directory. This collaborator is the **maintainer** of the repo, she has Write access to the repo and can decide which contributions are being included.

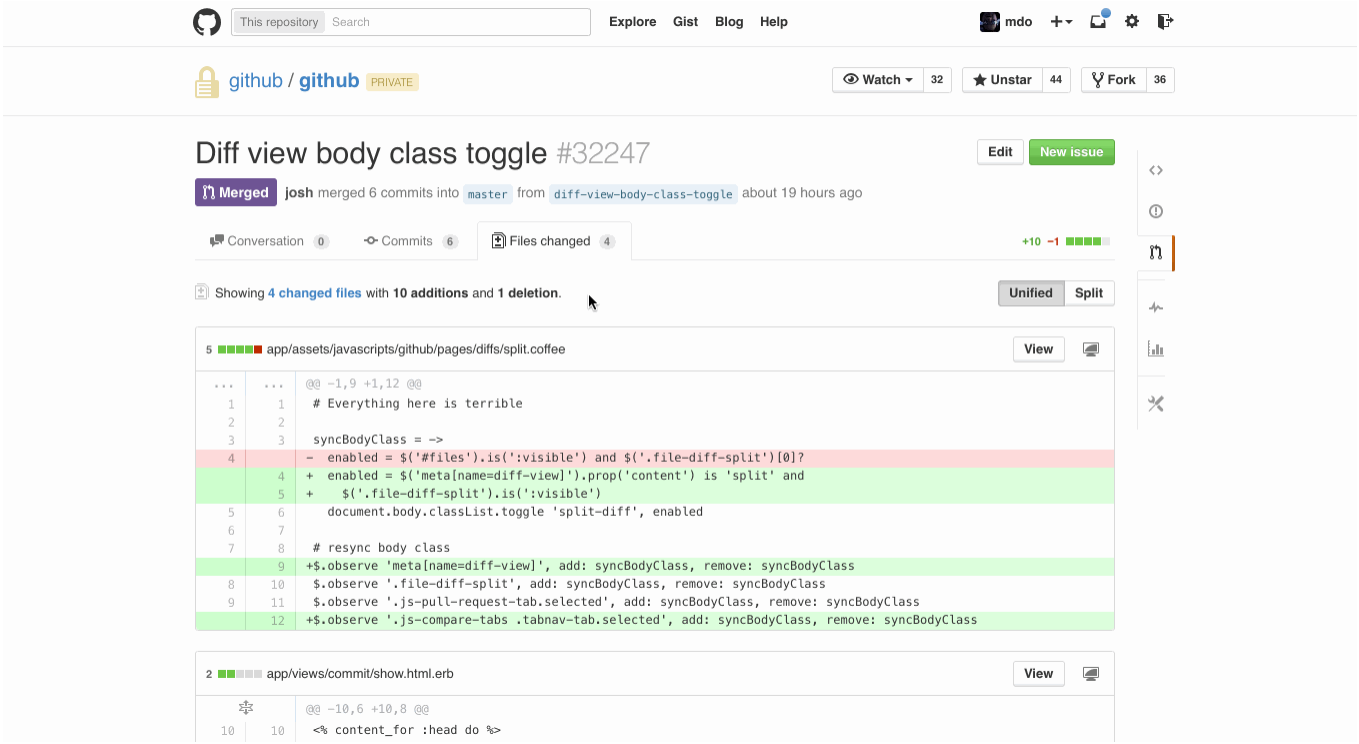
(2) Now, suppose the repository contains a file you want to modify. Instead of working directly in the repo, you will create a local **branch**. This branch is a copy of the repo, and you can modify it as you wish. Your modification will not affect the **master branch**. At least not yet.

(3) You do your work on your subsidiary branch, like modifying a .do file. When you save your modifications locally, you **commit** changes. The **commits** are basically checkpoint files: they contain a unique ID, the actual changes, and some metadata (time, author, commit message). The commits only store the *actual* changes you make to the text of a script, they do not save the whole branch. This is the reason why VC softwares can save long and intricate project histories, without making your system run out of memory.

Each commit is associated with a commit message. This is a way to give an overview of your changes to your collaborators.

(4) Once you are happy with your changes, you request a **pull** of your changes to the master branch. You are basically asking the maintainer to review, discuss and accept your work. We say that you **open a pull request**.

(5) The maintainer and you then discuss, and enact the changes. VC softwares offer user-friendly interface to discuss and review code. This is one of their main advantages. For instance, you can easily see the before and after versions of a script side by side, with helpful colours and annotations to highlight the changes. See an example of GitHub's tool 'split diffs' below:



Graph from the [GitHub Blog (https://github.blog/)](https://github.blog/)

(6) The last step consists in **merging** the branch you have worked on (and that you have discussed) with the master branch. When the maintainer decides to merge your branch to the master one, it becomes the new master branch, your changes are now the default. Note that pull requests have kept a track record of all the changes, so your collaborators can go back in time, reverting your changes as they wish (but on their own subsidiary branch!).

2.2. The difference between `Git` and GitHub

Steps (4) and (5) are specific to GitHub and other VC platforms, `Git` does not allow you to open pull request and communicate as easily with your co-workers. This highlight the difference between `Git` and GitHub: `Git` is simply the architecture of version control, while GitHub is a set of user-friendly, and productivity-enhancing features built on top of `Git` .

2.3. More complicated workflows

Version control gets more useful the bigger the team working on a project is, and the more intricate the different features of a project are. When engineers work for a client, for instance when building a website, the client may want to release some features of the website while asking the engineers to keep working on others. Using version control enables the client to cherry pick which features would need to go live, without worrying about the interdependencies. The client would simply need to go back in time to a stable version of the website which has all the desired features.

See an example of an agile workflow below:

References

Github Guide (https://guides.github.com/)

QuantEcon: Version control, git and github (https://lectures.quantecon.org/jl/version_control.html)

- Gentzkow, M. and J. M. Shapiro (2014). “Code and data for the social sciences: A practitioners guide”. *University of Chicago mimeo* link (https://web.stanford.edu/~gentzkow/research/CodeAndData.pdf) Chapter 3