1 Why Git? Why GitHub?

Why would a data analyst use hosted version control?

This intro has grown into a stand-alone article that is arguably a better introduction at this point. Until I merge it back in, consider reading the article instead: "Excuse me, do you have a moment to talk about version control?" https://dx.doi.org/10.7287%2Fpeerj.preprints.3159v2.

1.1 Why Git?

Git is a **version control system**. Its original purpose was to help groups of developers work collaboratively on big software projects. Git manages the evolution of a set of files – called a **repository** – in a sane, highly structured way. If you have no idea what I'm talking about, think of it as the "Track Changes" features from Microsoft Word on steroids.

Git has been re-purposed by the data science community. In addition to using it for source code, we use it to manage the motley collection of files that make up typical data analytical projects, which often consist of data, figures, reports, and, yes, source code.

A solo data analyst, working on a single computer, will benefit from adopting version control. But not nearly enough to justify the pain of installation and workflow upheaval. There are much easier ways to get versioned back ups of your files, if that's all you're worried about.

In my opinion, **for new users**, the pros of Git only outweigh the cons when you factor in the overhead of communicating and collaborating with other people. Who among us does not need to do that? Your life is much easier if this is baked into your workflow, as opposed to being a separate process that you dread or neglect.

1.2 Why GitHub?

This is where hosting services like GitHub, Bitbucket, and GitLab come in. They provide a home for your Git-based projects on the internet. If you have no idea what I'm talking about, think of it as DropBox but much, much better. The remote host acts as a distribution channel or clearinghouse for your Git-managed project. It allows other people to see your stuff, sync up with you, and perhaps even make changes. These hosting providers improve upon traditional Unix Git servers with well-designed web-based interfaces.

Even for private solo projects, it's a good idea to push your work to a remote location for peace of mind. Why? Because it's fairly easy to screw up your local Git repository, especially when you're new at this. The good news is that often only the Git infrastructure is borked up. Your files are just fine! Which makes your Git pickle all the more frustrating. There are official Git solutions to these problems, but they might require expertise and patience you can't access at 3a.m. If you've recently pushed your work to GitHub, it's easy to grab a fresh copy, patch things up with the changes that only exist locally, and get on with your life.

We target GitHub – not Bitbucket or GitLab – for the sake of specificity. However, all the big-picture principles and even some mechanics will carry over to these alternative hosting platforms.

Don't get too caught up on public versus private at this point. There are many ways to get private repositories from the major providers for low or no cost. Just get started and figure out if and how Git/GitHub is going to work for you! If you outgrow this arrangement, you can throw some combination of technical savvy and money at the problem. You can either pay for a higher level of service or self-host one of these platforms.

1.3 Is it going to hurt?

Yes.

You have to install Git, get local Git talking to GitHub, and make sure RStudio can talk to local Git (and, therefore, GitHub). This is one-time or once-per-computer pain.

For new or existing projects, you will:

- Dedicate a directory (a.k.a "folder") to it.
- Make it an RStudio Project.
- Make it a Git repository.
- Go about your usual business. But instead of only *saving* individual files, periodically you make a **commit**, which takes a multi-file snapshot of the entire project.

    - Have you ever versioned a file by adding your initials or the date? That is effectively a **commit**, albeit only for a single file: it is a version that is significant to you and that you might want to inspect or revert to later.
- Push commits to GitHub periodically.
    - This is like sharing a document with colleagues on DropBox or sending it out as an email attachment. It signals you're ready to make your work visible to others and invite comment or edits.

This is a change to your normal, daily workflow. It feels weird at first but quickly becomes second nature. FWIW, STAT 545 students are required to submit all coursework via GitHub. This is a major topic in class and office hours for the first two weeks. Then we practically never discuss it again.

More bad news. The STAT 545 pain is short-lived because students primarily work in their own repositories. Do you use GitHub to work with other people or to coordinate your own work from multiple computers? If so, after you recover from the initial setup, Git will crush you again with **merge conflicts**. And this is not one-time pain, this could be a dull ache for a long time. The best remedy is prevention, but also understanding how to back out of tricky situations and tackle them on your own terms.

The rest of this site is dedicated to walking you through the necessary setup and creating your first few Git projects. We conclude with prompts that guide you through some of the more advanced usage that makes all of this initial pain worthwhile.

1.4 What is the payoff?

Exposure: If someone needs to see your work or if you want them to try out your code, they can easily get it from GitHub. If they use Git, they can clone or fork your repository. If they don't use Git, they can still browse your project on GitHub like a normal website and even grab everything by downloading a zip archive.

Be a keener! If you care deeply about someone else's project, such as an R package you use heavily, you can track its development on GitHub. You can watch the repository to get notified of major activity. You can fork it to keep your own copy. You can modify your fork to add features or fix bugs and send them back to the owner as a proposed change.

Collaboration: If you need to collaborate on data analysis or code development, then everyone should use Git. Use GitHub as your clearinghouse: individuals work independently, then send work back to GitHub for reconciliation and transmission to the rest of the team. The advantage of Git/GitHub is highlighted by comparing these two ways of collaborating on a document:

- **Edit, save, attach.** In this workflow, everyone has one (or more!) copies of the document and they circulate via email attachment. Which one is "master"? Is it even possible to say? How do different versions relate to each other? How should versions be reconciled? If you want to see the current best version, how do you get it? All of this usually gets sorted out by social contract and a fairly manual process.
- **Google Doc.** In this workflow, there is only one copy of the document and it lives in the cloud. Anyone can access the most recent version on demand. Anyone can edit or comment or propose a change and this is immediately available to everyone else. Anyone can see who's been editing the document and, if disaster strikes, can revert to a previous version. A great deal of ambiguity and annoying reconciliation work has been designed away.

Managing a project via Git/GitHub is much more like the Google Doc scenario and enjoys many of the same advantages. It is definitely more complicated than collaborating on a Google Doc, but this puts you in the right mindset.

1.5 Who can do what?

A public repository is readable by the world. The owner can grant higher levels of permission to others, such as the ability to push commits.

A private repository is invisible to the world. The owner can grant read, write (push), or admin access to others.

There is also a formal notion of an organization, which can be useful for managing repository permissions for entire teams of people.

1.6 Special features of GitHub

this is perhaps too detailed … full stop? or does it belong elsewhere?

In addition to a well-designed user interface, GitHub offers two especially important features:

- **Issues.** Remember how we're high-jacking software development tools? Well, this is the bug tracker. It's a list of things … bugs, feature requests, to dos, whatever.
    - Issues are tightly integrated with email and therefore allow you to copy/embed important conversations in the associated repo.
    - Issues can be assigned to people (e.g., to dos) and tagged ("bug" or "progress-report").
    - Issues are tightly integrated with commits and therefore allow you to record *that the changes in this commit solve that problem which was discussed in that issue*.
    - As a new user of GitHub, one of the most productive things you can do is to use GitHub issues to provide a clear bug report or feature request for a package you use.
- **Pull requests.** Git allows a project to have multiple, independent branches of development, with the notion that some should eventually be merged back into the main development branch. These are technical Git terms but hopefully also make sense on their own. A pull request is a formal proposal that says: "Here are some changes I would like to make." It might be linked to a specific issue: "Related to #14." or "Fixes #56". GitHub facilitates and preserves the discussion of the proposal, holistically and line-by-line.

1.7 What's special about using R with Git and GitHub?

- The active R package development community on GitHub. Read about R-specific GitHub resources and searching here.
- Specific workflows make it rewarding to share source code, rendered reports, and entire projects. Read more about R Markdown, R scripts, and R-heavy projects.
- Git- and GitHub-related features of the RStudio IDE. This is covered throughout.

1.8 Audience and pre-reqs

The target audience for this site is someone who analyzes data, probably with R, though some of the content may be useful to analysts using other languages. R package development with Git(Hub) is absolutely in scope, but it is not an explicit focus or requirement.

The site is aimed at intermediate to advanced R users, who are comfortable writing R scripts and managing R projects. You should have a good grasp of files and directories and be generally knowledgeable about where things live on your computer.

Although we will show alternatives for most Git operations, we will inevitably spend some time in the shell and we assume some prior experience. For example, you should know how to open up a shell, navigate to a certain directory, and list the files there. You should be comfortable using shell commands to view/move/rename files and to work with your command history.

1.9 What this is NOT

We aim to teach novices about Git on a strict “need to know” basis. Git was built to manage development of the Linux kernel, which is probably very different from what you do. Most people need a small subset of Git’s functionality and that will be our focus. If you want a full-blown exposition of Git as a directed acyclic graph or a treatise on the Git-Flow branching strategy, you will be sad.