

# Running a reproducible analysis

H2L2C, Project Management Day 1




AUTHOR  
Brian Gural

## Reproducible science... in-silco??

- Bioinformaticians are people too
- We need to make sure our research is well documented and reproducible just like bench scientists
- Projects can get complex, messy, and very computationally demanding

## How can computational projects get derailed?

It turns out that computational biologists need to be careful with how they manage their code and data. Leaving everything on your personal/lab computer comes with a lot of risk 

You can reduce the risk of a mishap by housing data on UNC's cloud computing service, **Longleaf**, and putting your code on **Github**. Both of these provide you with backups that can be accessed from anywhere with the internet.

Don't think it's worth it? Here are some moments that made UNC researchers wished they had used these tools:

Broken laptops, crushed dreams

"How did you make this figure from 2018?"

"...there was a time where my computer just stopped letting me log in and needed to be wiped so if I wasn't using Longleaf I would have lost everything. I did lose a nice powerpoint."

"In undergrad I was using local storage only on the desktop in my advisor's office. There was some big failure with IT one day (tbh I still don't know what happened) and I lost all my code"

"Our collaborator lost the hard drive with the raw RNAseq data, dooming my first 1st author publication. His collaborator saved the day with a backup he had on Longleaf"



## Good computational practices 101

Computational projects ought to be approached with the same expectations of rigor and reproducibility expected of a bench project. This means that the work needs to be well documented, things need to be properly stored, and everything should be organized clearly enough for someone to reproduce it.

Thankfully, we're not the first researchers to run into these problems. A whole suite of tools and services exist to manage these issues:

- Documenting everything [git](#) /GitHub
- Storing data & getting resources Longleaf
- Keeping your R project organized RStudio Projects

## Suffering from manual version control? [git](#) can help.

What is **version control** exactly? At its core, it's a way of keeping track of the changes made to files. If you've ever worked on a publication with a picky professor, you've probably tried to keep some kind of **home-brewed version control system** yourself by changing up the file names between drafts, something like this:

Before GitHub

[After GitHub](#)

```
paper_draft1.doc
paper_draft2.doc
paper_reviewed_by_john.doc
paper_draft3_comments_incorporated.doc
paper_final_draft.doc
paper_final_reviewed.doc
paper_final_submission.doc
paper_final_submission_revised.doc
paper_final_submission_revised_v2.doc
paper_published_version.doc
```

Essentially, the authors are trying to use the file name to keep track of the changes. But with a dedicated version control system, like [git](#), **you can update a file while keeping a detailed log of the changes.**

## Go on, [git](#)!

[git](#) is version control system used to record changes to files. GitHub uses [git](#) to help users host/review code and manage projects

[git](#) /GitHub matter because they:

- Track every version of every script
- Publicly document your work
- Allow for new versions of projects to [branch](#)

- Make it easy to collaborate

## Longleaf: The darling of UNC bioinformaticians

Longleaf is UNC's high-performance computing cluster (HPC). It's basically a ton of computers/storage. Its accessible from anywhere with internet and offers a lot of storage. Labs typically start with 40 TB, users get 10 TB. Also you've been using it this whole time! RStudio OnDemand is hosted by Longleaf.

### Tip

There are a ton of reasons to use LL:

- Many scripts can be run at once, with your computer off
- It has A LOT more resources than a typical computer
- Easy to share files!
- Dedicated technical support via ITS

## Connecting Longleaf and Github

Github and Longleaf each can be daunting to novice programmers, so lets walk through how to set them up together. 

The setup is going to amount to three general steps:

1. Introduce our Github and Longleaf accounts to each other with something called a **SSH** key
  - SSH keys (**secure shell**) are encrypted passwords that link two computer systems
2. Make our first repository (project) on Github
3. Learn how to get scripts from Github to Longleaf and update changes we made on Longleaf back to Github

Before we can start that, we're going to need to know just a tad about **terminals** and **Bash**. These topics could be a whole course onto itself, but in a nutshell you can think of them like this:

**Terminals**, also called **command lines**, are text-based software for interacting with your computer. RStudio has a built-in terminal, on the tab next to "Console".

**Bash** is a type of computer language that understands and carries out the instructions you type in the terminal, usually called "shell scripting". It's very common on Linux and Mac computers. Longleaf uses Linux and working on Longleaf means using a bit of Bash.

### Tip

This course isn't meant to teach you shell scripting, and you aren't expected to fully understand some of the Bash command we'll run. If you'd like to learn more XYZ are great resources!

# Linking via SSH

---

Let's start by getting the terminal open. In the top left, click View -> Move Focus To Terminal. It should've opened in the panel that also contains tabs for "Console" and "Background Jobs".

Next, let's find the SSH key associated with your Longleaf account. We'll run the following bash command in the terminal that we just opened (be aware that terminals are notoriously finicky with copy and paste): `cat ~/.ssh/id_rsa.pub`

This (hopefully) has copied your SSH key to your clipboard!

Now, let's go over to Github and set up the key.

1. Go to profile settings on github and select the "ssh key" section
2. Add new key
3. Name the key (should remind you that this is the key for Longleaf)
4. Paste the copied ssh key from the `cat` step above
5. Create!

With that done, we need to log into Github on Longleaf:

1. Go back to the RStudio terminal
2. `git config --global user.name "your-github-username"`
  - Keep the quotes around your username!
3. `git config --global user.email your.email.linkedwithgithub`
  - No quotes this time!

## Making a repository

---

Great, we've connected Longleaf and Github (a herculean task for a beginner programmer!). What we'll want to do next is make a repository (repo), which you could think of as a self-contained project folder. Let's go back to Github:

1. In the "Repositories" tab of your profile, click "New"
2. Give it a name, maybe "example\_repo"
3. Click the "Add a README file box"
4. Add a `.gitignore`` with an R template
5. Create the repo!

We'll explain the significance of the README and .gitignore steps a bit later. For now, let's go over to our new repo on our profile. To get it onto Longleaf, we can:

1. Click on the green "Code" box
2. Click on "SSH"
3. Copy the SSH right below that! It should end in ``.git``
4. Go back to the terminal in RStudio and run `git clone` followed by the whole link you just copied.
5. !!!!!!!!!!!!!!! DECIDE WHERE WE WANT THEM TO MAKE THE REPO
  - It should look like this: `git clone URL-THAT-YOU-COPIED`

## RProjects:

---

## Using git

---

- `.gitignore` uses rules to exclude files by name pattern or location
  - Don't upload data or large files!
- Changes need to be staged with `git add`
  - `git add .` adds all files not excluded by the .gitignore in the directory
  - `git add -i` opens an interactive adding session
- Commit staged changes with a note to your future self
  - `git commit -m "Hi future me, this is what I changed"`
- Commits are pushed to branches
  - For the main branch, use `git push origin main`