Collaborative & Reproducible Research

Lesson 3 with *Ian Carroll*

Collaboration & Code

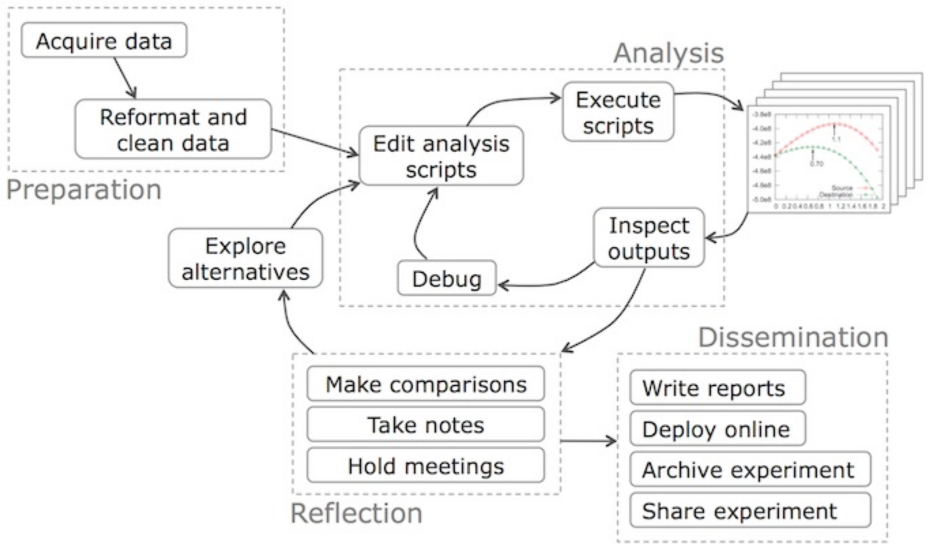
As your research project moves from conception, through data collection, modeling and analysis, to publishing and other forms of dissemination, it's components can fracture, lose their development history, and—worst of all—become conflicted or lost. This lesson introduces strategies for structuring your collaboration, focusing on the core principle of "version control" (VC) and the related software and cloud solutions. The main objective is to demonstrate that version control is much more than history, it is how distributed teams can work efficiently on a shared codebase and is widespread in reproducible research.

A cloud **hub** stores project files and their history. Researchers are spokes on the wheel, each working on a local **repository**. Project integrity is maintained by rules for synchronizing **commits** between the hub and spokes when users execute a **push** or **pull**.

Top of Section

Lesson Objectives

By cultivating new habits for scripting, documenting, and committing, much of the research cycle can be made reproducible.



Credit: *Philip Guo*

- Learn about software repositories
- See that VC equals efficient communication
- Identify attributes of reproducible research
- See how RStudio + git facilitate collaboration

Specific Achievements

- Make "commits" to a repository with git
- "Push" and "pull" project work to GitHub
- "Merge" your work with a collaborator's via GitHub

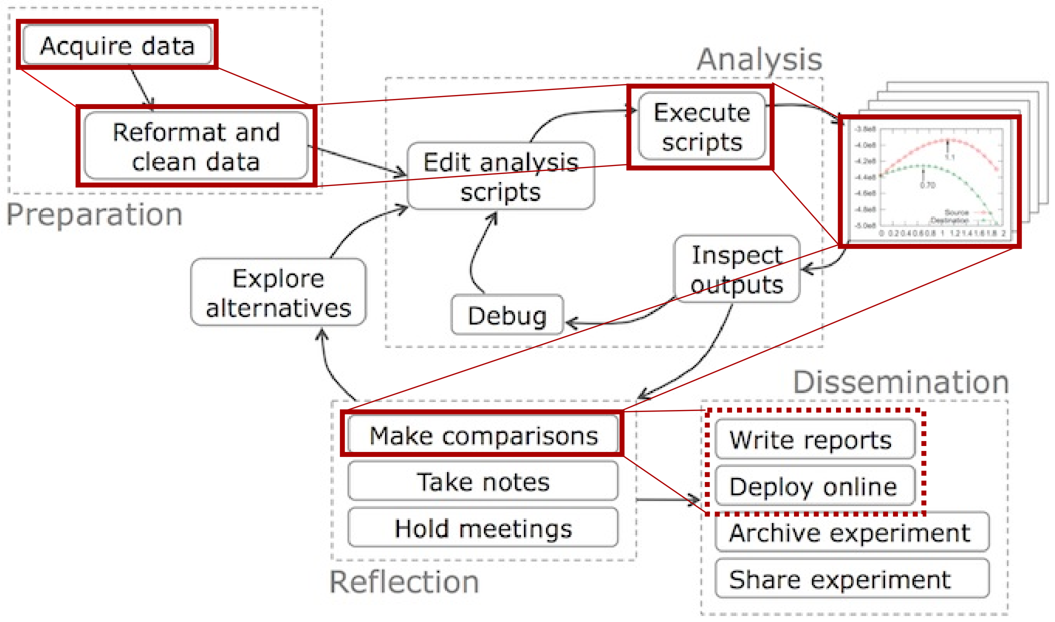
Top of Section

Reproducible Pipeline

The result of reproducible research is more than a published paper, it includes the whole data-to-document **pipeline**. In a typical socio-environmental synthesis project, a finished pipeline ideally includes the following steps:

1. Aquire raw data from online repository.
2. Extract, transform, and load data into storage for analysis.
3. Perform data analysis (e.g. model inference) and visualization.
4. Update documentation and reports.
5. Publish results, including reports, data and code/software.

The pipeline is a direct path through the research cycle, avoiding all its twists and turns, so others may reproduce your final analysis.



Credit: Philip Guo

A UC Berkeley professor who is a strong advocate for open science, Carl Boettiger, has released several reproducible pipelines on GitHub. For example, check out his work leading up to the paper "Pretty Darn Good Control" in the project pdg_control.

Top of Section

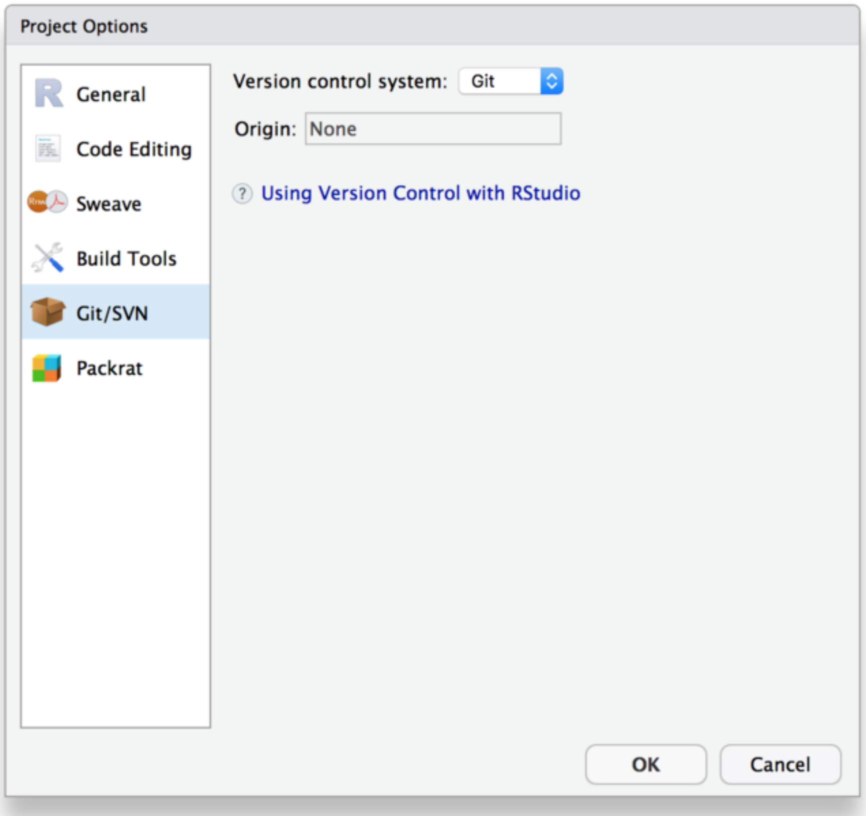
RStudio + git

Login to your RStudio Server account and upload `handouts.zip`. Click on `handouts/handouts.Rproj` to open the directory as a project.

RStudio provides a GUI to the core tools provided by git. Every folder that contains a "*.Rproj" file is an RStudio project (simply a collection of files and configuration settings). We use `git` to turn our project into a "repository", a term from the software development community that encompases a project along with its development history.

Initialize git

Convert your RStudio project to a git repository by enabling version control, available from the menu bar under "Tools" > "Version Control" > "Project Setup".



Version control system: Git

Origin: None

Using Version Control with RStudio

Adding a git repository creates a hidden folder in your project called ".git", storing all the data about your project's current and past state.

Commit

Once RStudio refreshes your project, there will be a "Git" tab in the same window as the Environment tab. The window shows files that have content not already commited in the current state of your project. Choose "Commit" to open a new window for easy staging and commiting.

1. check `README.md` and `handouts.Rproj`
2. write a commit message
3. commit (but heed the warning!)

Saving, staging, and commiting are each separate steps, none of which imply any of the others. This may seem like a hassle, but is a good thing! As your project grows larger, you will frequently save changes you don't want to commit: staging lets you choose what changes get packaged into a commit.

History

The history of your project shows a single commit, every new commit will be chained on top of a preceding commit. Note the "Author" data is probably not going to be recognized by GitHub and linked to your account.

For GitHub to associate commits with your account, configure git with your GitHub username and email address.

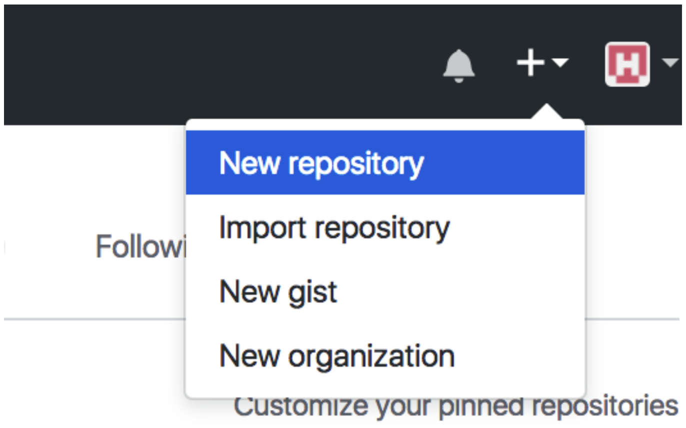
worksheet.R

```
git config --global user.name jdoe
git config --global user.email jdoe@sesync.org
git commit --no-edit --amend --reset-author
```

Revisit the commit history to confirm that the author information has been amended for the first commit. In the future, configure your `user.name` and `user.email` before starting a project, so you do not have to ammend any commits. This is a one-time configuration for each computer on which you use RStudio, so you don't have to repeat this for any subsequent repositories you create on this machine.

Create the Hub

Sign in or sign up on GitHub, then create an empty repository.



1. Give your repository the same name as your RStudio project.
2. Add a short “tag line” about your workshop experience.
3. **Do not check either box.**

You have created a repository that has no history—it will accept the commits made in RStudio without conflict. The quick start information provided by GitHub explains how to finish configuration of your local git repo.

Console

```
git remote add origin https://github.com/jdoe/handouts.git
git push -u origin master
```

Go back to your GitHub account and check out your “hub”.

- `README.md` is a Markdown file giving basic information about the repository.
- There is a list of files, including a folder for data.
- You are looking at a branch called `master`.
- The commit history is available from the top bar.
- The “Clone or download” button provides a URL.

In addition to being the center point for sharing commits with collaborators, GitHub is a rich platform for managing projects and inspecting the history.

Top of Section

Syncing Repos

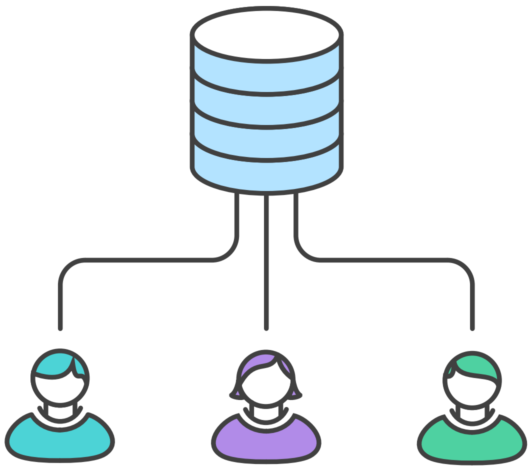


Image by Atlassian / CC BY

The **origin** is the central repository, in this case it lives on GitHub. Every member of the team gets a **local** copy of the entire project, called a **clone**.

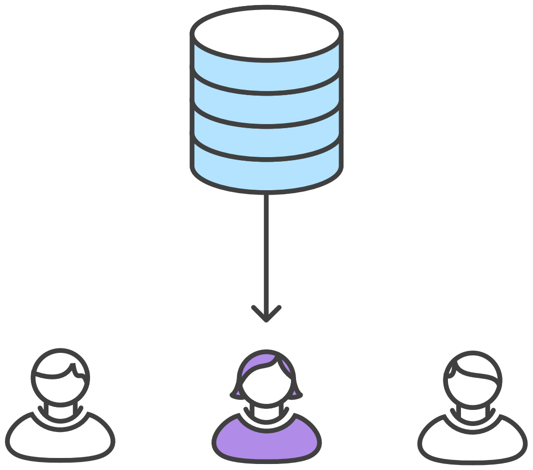


Image by Atlassian / CC BY

Cloning is the initial **pull** of the entire project and all its history. In general, a worker **pulls** the work of other teammates from the **origin** when ready to incorporate their work, and she **pushes** updates to the **origin** when ready to contribute work of her own work.

A git repository is a network of commits, although the current network is a tree with no splits. After a worker creates a **clone**, the local repo is in the same state as the **origin**.

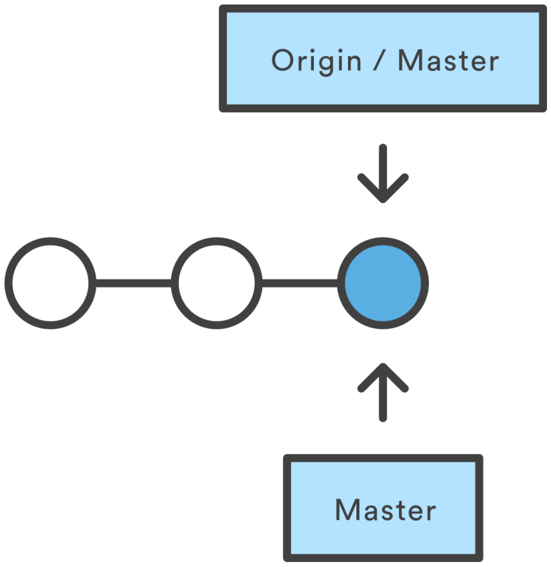


Image by Atlassian / CC BY

When the origin has commits that do not exist in the local repo, it has gotten ahead and a **pull** is required to synchronize state.

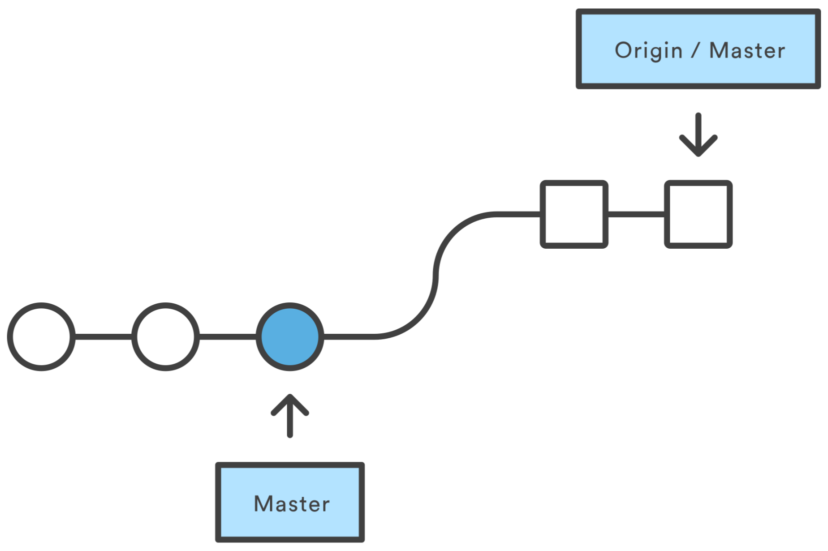


Image by Atlassian / CC BY

A **pull**, or initially a **clone**, applies commits copied from the **origin** to your local repo, as if you had created identical commits locally.

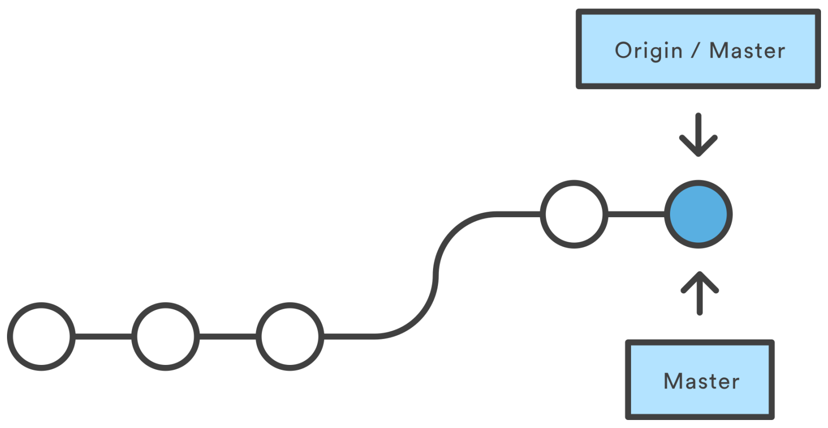


Image by Atlassian / CC BY

In the opposite situation, commits created locally are not immediately synchronized to the **origin**.

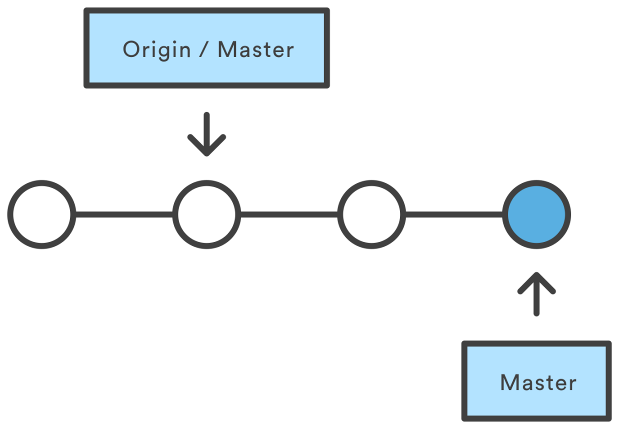


Image by Atlassian / CC BY

A **push** copies local commits to the **origin** and applies them remotely.

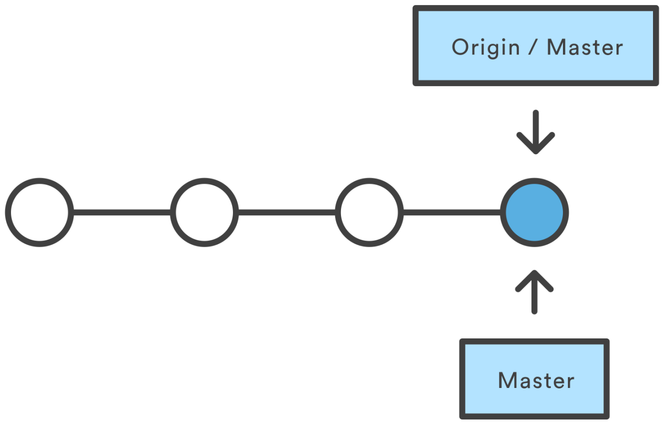


Image by Atlassian / CC BY

An essential component of version control is the ability to merge commit histories that have diverged.

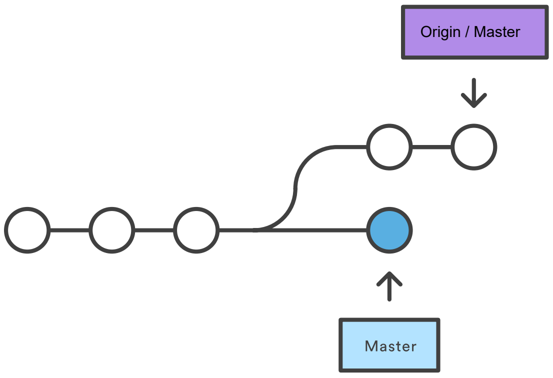


Image by Atlassian / CC BY

The commit graph splits any time different commits are applied to the same “parent” commit. Automatic merging done by git integrates the changes from both into a new “merge commit””, unsplitting the commit graph.

The **origin** will not accept a push that requires merging. In order to preserve integrity, the contributor is always responsible for overseeing the merge on a local **clone**.

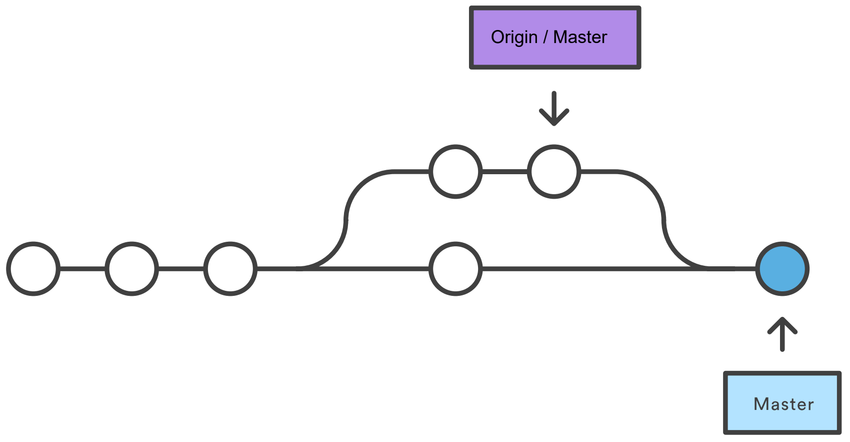


Image by Atlassian / CC BY

After the merge commit has been created locally, the same situtation now exists that was depicted above for a **push** that the origin will accept.

Top of Section

Collaborators

Collaboration that goes beyond commenting on a final report—integrated work on a project from start to finish—raises serious challenges. Distributed repositories, managed by git, help to answer these questions.

- Data, script, or report; who has the latest version?
- Will a collaborator's work erase or break your own?
- How to recover a working version of a pipeline?

Project Integrity

- The origin becomes the official "latest" version, even if *your* work is a few commits ahead.
- Diverging files are usually automatically merged by git.
- Manual re-integration is aided by the ability to "checkout" the project at any commit.

Version control software works well with text files. Large, non-text components of your project (e.g. very big or binary data files) can slow down any cloning, merging or branch switching. For that reason, data rarely live in a repository with code and. Keeping data and code separate also facilitates data reuse—it's not tied to one pipeline.

Add a section where you can list collaborators to the README.md file. Our aim is to let your collaborators update this list with their own name, so only include yourself. You can use any text editor, and RStudio's is quite handy.

README.md

```
## Collaborators

- J. Doe
```

Stage

Before you can commit changes involving a new file, you have to tell git which modifications you want to commit by staging.

1. Go to the "Git" tab in RStudio.
2. Select "Commit" to open the "Review Changes" window.
3. Select "Staged" to add modifications (hence "M") by "README.md".

Commit

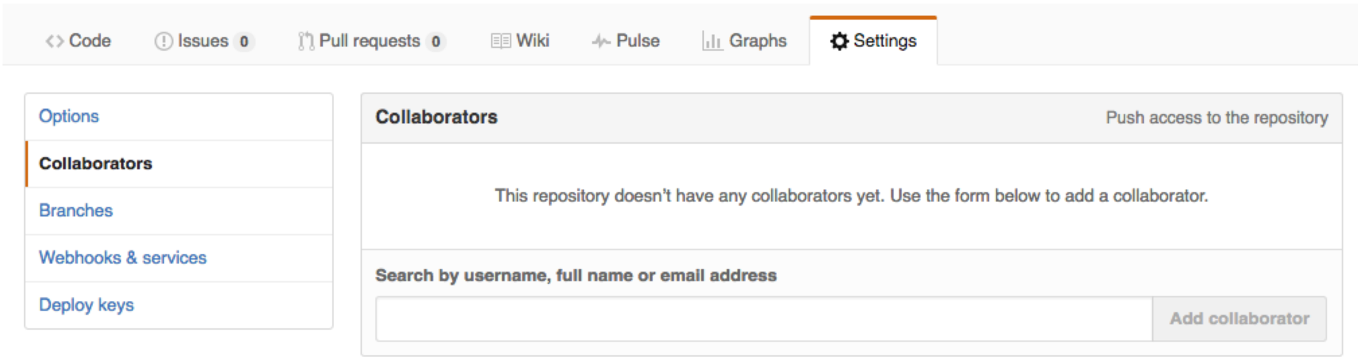
1. Enter a brief (<50 chars) descriptive message about the commit.
2. Commit!
3. Close the “Review Changes” window.

Push

Look at the “Git” tab again and notice that your branch is “ahead of origin/master”. Push the commit to your GitHub repo.

GitHub Collaborators

Even on public GitHub repos, only the owner has “push access” by default. The owner can allow any other GitHub user to push by inviting collaborators under the settings tab.



This repository doesn't have any collaborators yet. Use the form below to add a collaborator.

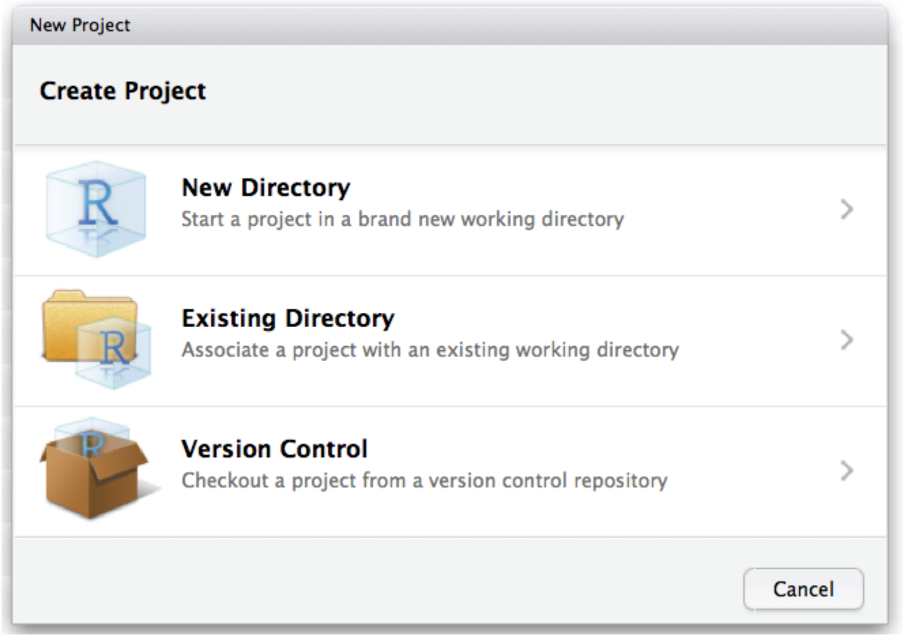
Introduce yourself to your neighbor and assign split the two roles below between you. Be sure to watch eachother perform the steps assigned to your individual roles.

1. Owner: add your neighbor as a collaborator.
2. Collaborator: accept your neighbor’s emailed invitation.

Create a Second Spoke

The **Collaborator** now needs to create a new project in RStudio by cloning the **Owner’s** project. Under the “File” menu item, choose to create a New Project, and then choose “Version Control”.

You cannot use the same name for two project folders! The **Collaborator** should chose a different name for their copy of the **Owner’s** project.


New Project
Create Project
New Directory
Start a project in a brand new working directory
Existing Directory
Associate a project with an existing working directory
Version Control
Checkout a project from a version control repository
Cancel

Push & Pull

1. Collaborator: Add your name to the list in the “README.md”.

2. Collaborator: Stage, commit, and push (up arrow) your modifications.
3. Owner: Pull (down arrow) to apply your neighbor’s commit.

Merge

You both realize it would have been good to include your affiliation along with your name. Do you need to circulate “README.md” to each collaborator in sequence for an update? No!

1. Ower **AND** Collaborator: edit your entry in the “README.md”
2. Ower **AND** Collaborator: stage, commit, & push.
3. Owner **OR** Collaborator: if you receive an error message, it tells you exactly what to do.

After successfully complete a merge, move back to your own “handouts” project in RStudio to access your own worksheets.

Top of Section

What about Data?

The scripts tha execute your pipeline are plain text files, but the project may need other assets, such as data, figures, or private configurations.

- Non-text files get little benefit from `git`, and have large costs.
- Large data files should not be version controlled by `git` and usually live outside the repo (as an “integration”).
- Private information (e.g. personal data) should not be committed to a public repository.

External Data

The most common pipeline integration is shared data storage.

- Local area network file share (e.g. “Z:\\…”)
- Cloud storage (e.g. Dropbox, Google Drive)
- Database (e.g. a PostgreSQL server)

Link to the Data

One good practice is creating “symbolic links” (a.k.a. shortcuts) to data files that live outside a project repo; these let your code use paths that point inside the repo for data.

worksheet.R

```
file.symlink(
  from = ...,
  to = 'data'
)
```

The shortcut works like a normal path to your data—which could be risky on certain operating systems or early versions (before 1.6) of “git”. It is sometimes possible to add all your data to a commit by accident with `git add .`. To avoid this, update git or “ignore” all files and folders below `data/` by adding the line `/data/**` to the “.gitignore” file in your repo. The leading `/` refers to the root of the git repository, not to the root of your filesystem.

Top of Section

A Plug for Reproducibility

Reproducibility is a core tenent of the scientific method. Experiments are reported in sufficient detail for a skilled practitioner to duplicate the result.

The principle equally applies to modeling and data analysis.

Hallmarks of Reproducibility

Reviewable	All details of the method used are easily accessible for peer review *and* community review.
Auditable	[Private] records exist to document how the methods and conclusions evolved.

Replicable	Given sufficient resources, a skilled practitioner could duplicate the research without any guesswork.
Open	The orginator grants permissions for reuse and extension of the research products.

This table is adapted from “Best practices for computational science: Software infrastructure and environments for reproducible and extensible research” by V. Stodden and S. Miguez (2013).

Habits to Cultivate

Reviewable	Thoroughly-comment scripts and share continuously with collaborators
Auditable	Maintain project history to correct mistakes when necessary
Replicable	Provide “one-click” file & data sharing, of a streamlined analysis “pipeline”
Open	Publically release on GitHub (or similar) with (implied) open licensing

Learning to use git will take care of much of the work needed to make your research reproducible. Once you become efficient at staging, commiting, pushing and pulling, it will become the natural way to communicate with your collleagues through the process of collaborative development of a analysis pipeline.

Top of Section

Future Directions

- Learn more git: merge conflicts, reversion, and branching!
- Share your work for reuse and extension.
- Make trying new analysis as easy as branching.
- Contribute beyond your own projects.

The repository you created is an example of the heart of a distributed workflow. Putting the **origin** of your project on GitHub (or similar) will make it accessible not only by your collaborators, but also available for review and extension by your research community.

Using advanced `git` to manage contributions to the project as a branching and merging “tree” of commits accomplishes two objectives. First, work can safely proceed in parallel, on separate branches if necessary. Second, a recoverable (and auditable) trail of changes is immediately available in the project history.

The latest software for modeling and analysis in your research field may already be on git. Build better pipelines by contributing bug reports or even pull requests to projects integral to your own work.

Top of Section

Exercises

Exercise 1

Make the remaining worksheets show up in your “handouts” repository on GitHub. Remember the three steps: stage, commit, and push.

Exercise 2

Repeat the steps you walked through in pairs, as an **Owner** and **Collaborator**, but reverse the roles.

Top of Section

If you need to catch-up before a section of code will work, just squish it's 🍅 to copy code above it into your clipboard. Then paste into your interpreter's console, run, and you'll be ready to start in on that section. Code copied by both 🍅 and 📋 will also appear below, where you can edit first, and then copy, paste, and run again.

```
# Nothing here yet!
```