

# Version Control

Marco Morales  
[marco.morales@columbia.edu](mailto:marco.morales@columbia.edu)

GR5069  
Topics in Applied Data Science  
for Social Scientists

Spring 2019  
Columbia University

# RECAP: a Data Science project

Three **aims** of a Data Science project

a) **reproducibility**

- ▶ anyone should be able to arrive to your **same results**

b) **portability**

- ▶ anyone should be able to **pick up where you left off** on any machine

c) **scalability**

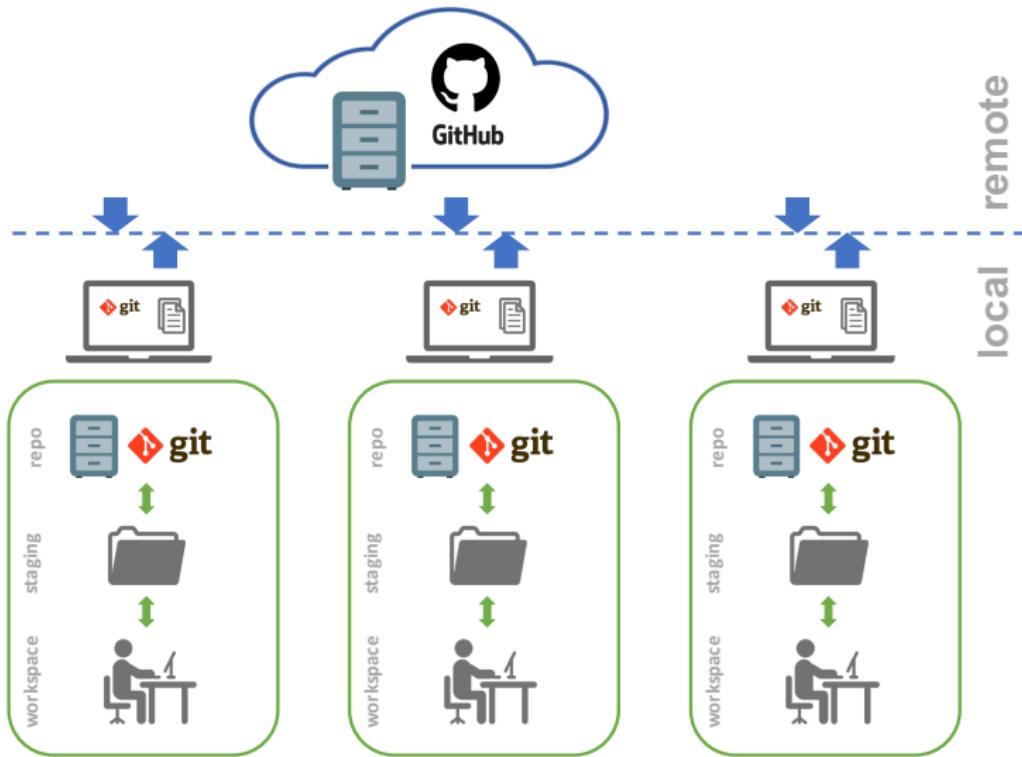
- ▶ your project should also work for **larger data sets** and/or be on the path of **automation**

a) and b) crucial for **collaborative work**

## RECAP: Version control

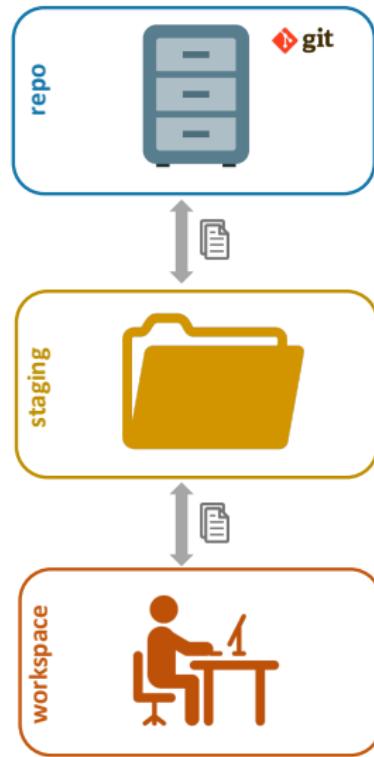
- ▶ **version control** allows you to keep track of changes / progress in your code
  - ▶ keeps “**snapshots**” of your code over time
  - ▶ helpful to **debug**, and to enhance **reproducibility**
  - ▶ also great for **team collaboration** (everyone can see who changed what!)
- ▶ **Git** is a version control software
- ▶ **GitHub** is an online open source repository

# An ideal version control setup for collaboration



# git locally

# git: a mental model



# Introduce yourselves: git, meet your new user!

from the command line:

- ▶ set your **user name** and **email address**

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email johndoe@example.com
```

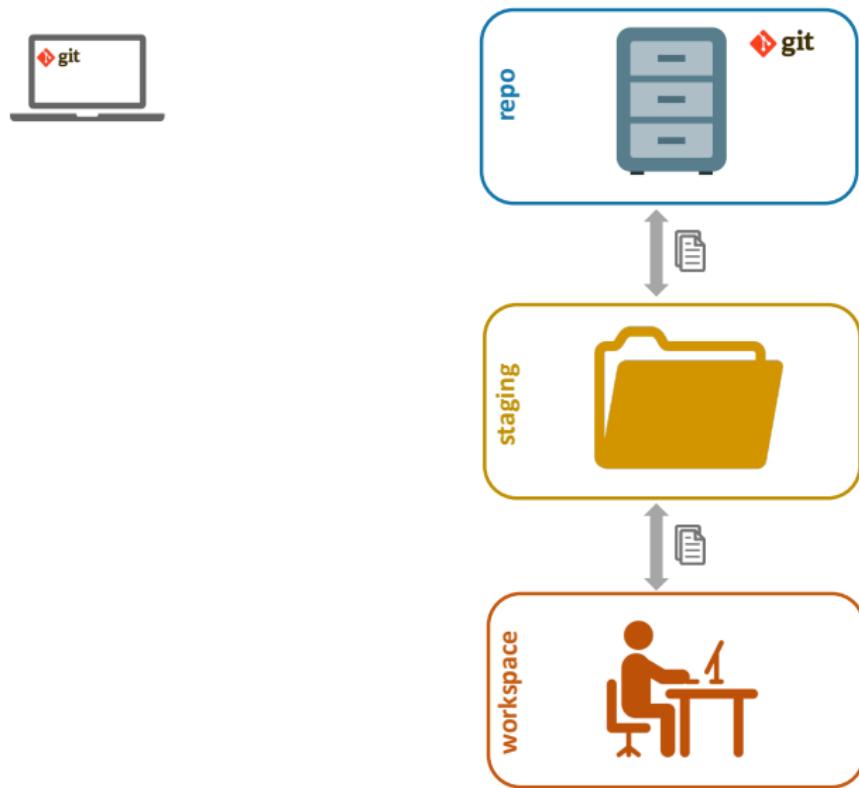
- ▶ **verify** that information was successfully entered

```
$ git config --list
```

- ▶ this information gets baked in your commits

- ▶ **ProTip:** other useful information (e.g. proxy settings) also goes on `git config`

# now, turn your folder structure into a git repo



# now, turn your folder structure into a git repo

from the command line:

- ▶ go to the **root** of your project and **initialize** the repo

```
$ git init
```

- ▶ there are **files you never want tracked** by git (e.g. log files, access keys), even by mistake

- ▶ from the root of your local repository, create a `.gitignore` file

```
$ touch .gitignore
```

- ▶ add files you want git to ignore in the `.gitignore` file

# what could go into a .gitignore file ?

```
# OS generated files #
*.DS_Store

# History files
*.Rhistory
*.Rapp.history

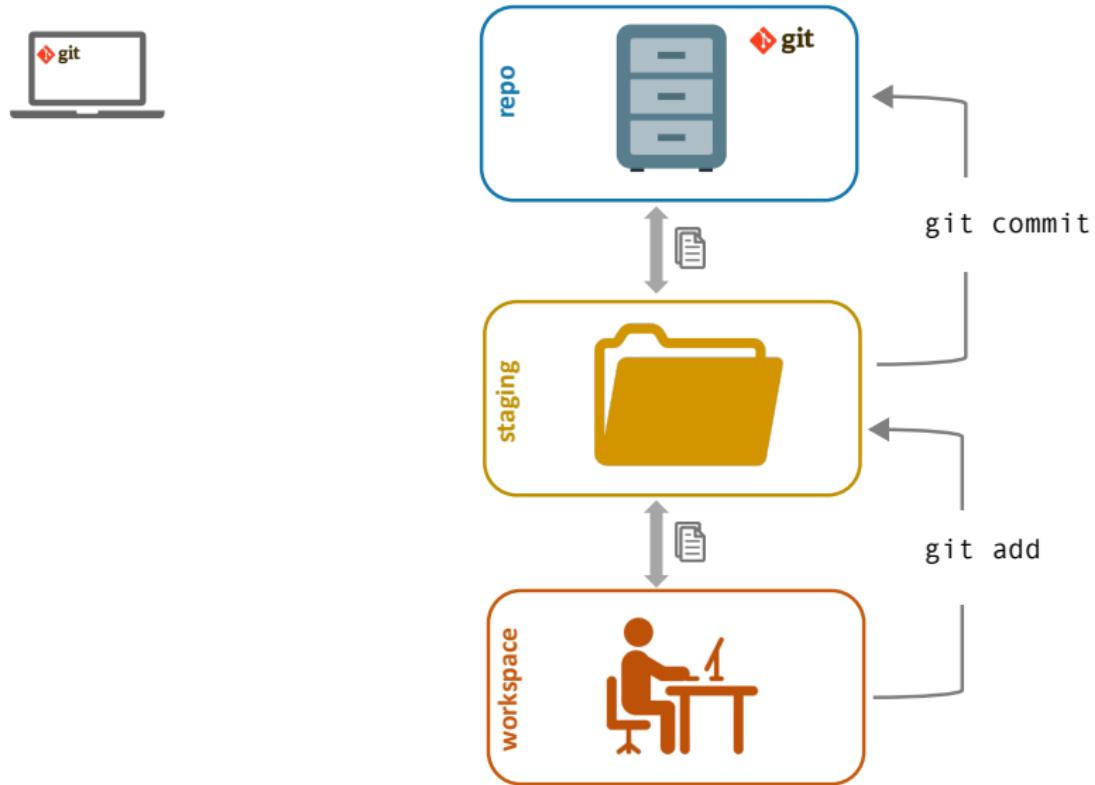
# Session Data files
*.RData

# RStudio files
*.Rproj.user/

# all data folders
data/
```

- ▶ **ProTip:** further info/templates:  
<https://github.com/github/gitignore>

# your basic git workflow



# your basic git workflow

from the command line:

- ▶ indicate a file to be tracked by git

```
$ git add samplefile.R
```

- ▶ verify what's being tracked

```
$ git status
```

- ▶ commit your tracked files (with an informative message)

```
$ git commit -m "Commit initial files"
```

## a few confusing things about git

- ▶ a file will be committed **exactly** as it was when you `git add`-ed it
- ▶ if you change the file **after** you `git add` it, and want to commit the new changes, you need to `git add` again before the `git commit`
- ▶ use `git status` to assess what's being staged and will be committed

# git workflow ProTips

- ▶ **NEVER** use `git add .`
- ▶ use `git status` often as **validation**
- ▶ only add and commit **source files**
  - ▶ omit files you can reproduce using source files
- ▶ commit **small chunks of logically grouped changes**
  - ▶ you may want to undo a change, and only that change
- ▶ commit with **informative** (imperative mood) **messages**
  - ▶ *[this commit will]* Rename income variable

**push globally  
(to GitHub)**

# first, create a GitHub repo to store/share in the cloud

## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner



marco-morales ▾

Repository name

testrepo



Great repository names are short and memorable. Need inspiration? How about friendly-octo-guide.

Description (optional)

a test repository



Public

Anyone can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

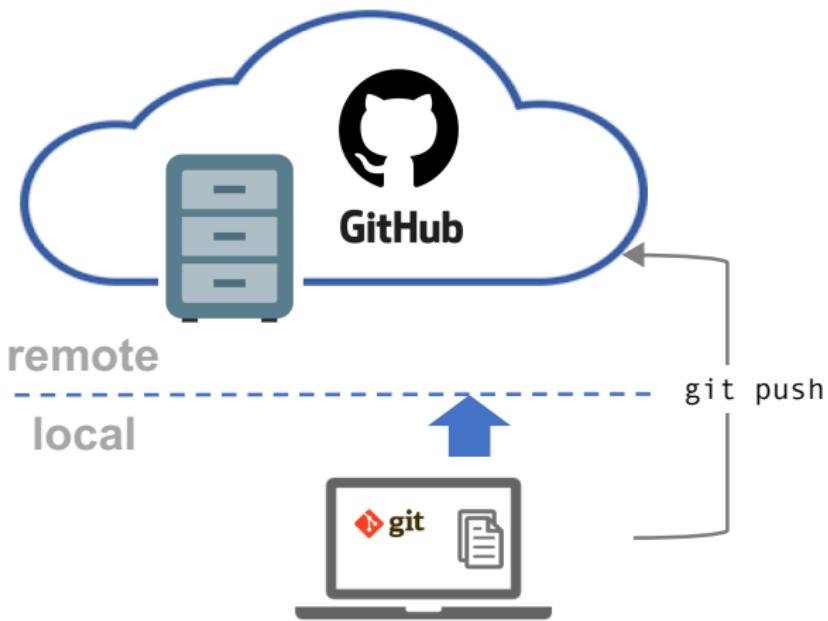
Add .gitignore: None ▾

Add a license: None ▾



**Create repository**

then, push to that GitHub repo



## then, push to that GitHub repo

from the command line:

- ▶ tell git the **location** of the remote GitHub repo you just created (typically nicknamed “origin”)

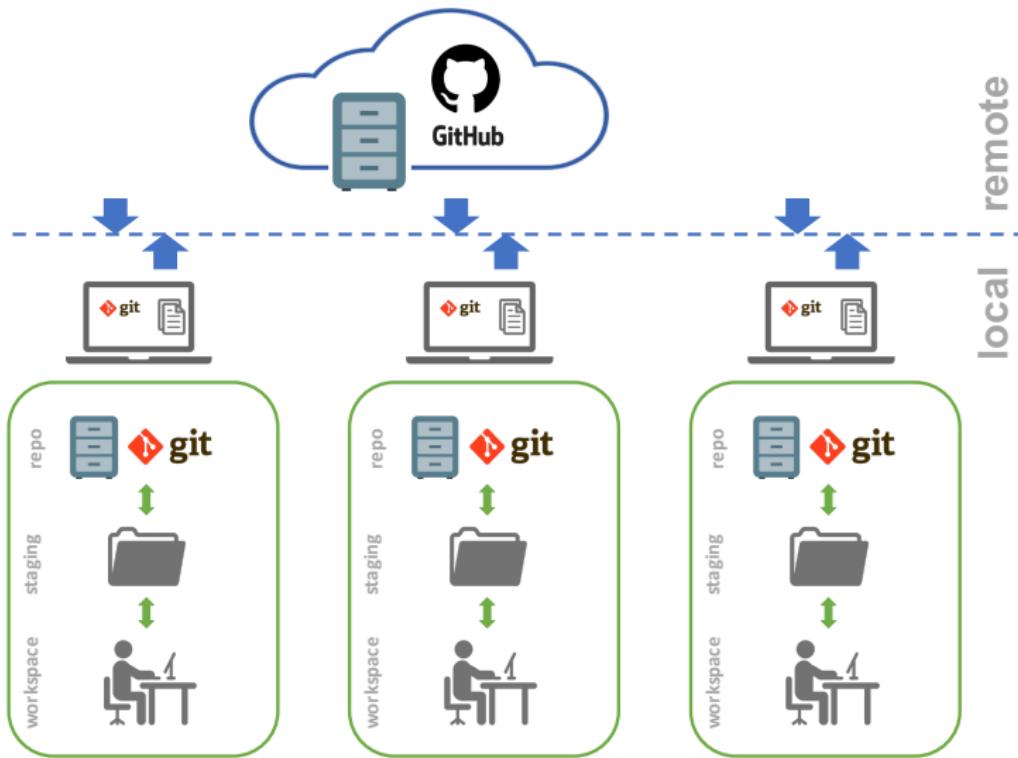
```
$ git remote add origin  
https://github.com/marco-morales/testrepo.git
```

- ▶ send **committed files** from your local git (“master”) repo to your Github (“origin”) repo

```
$ git push -u origin master
```

# git+GitHub for team collaboration

# all the building blocks are now in place



# now, enable collaborators in your GitHub repo

marco-morales / **testrepo**    Unwatch ▾ 2    Star 0    Fork 0

Code Issues 0 Pull requests 1 Projects 1 Wiki Insights Settings

**Options**

**Collaborators**

Branches

Webhooks

Notifications

Integrations & services

Deploy keys

Moderation

Interaction limits

**Collaborators** Push access to the repository

gulbzrhn 

Search by username, full name or email address  
You'll only be able to find a GitHub user by their email address if they've chosen to list it publicly. Otherwise, use their username instead.

gulbzrhn **Add collaborator**

# important to know what each role can do

- ▶ add **collaborators** to your repo
  - ▶ as a repo **owner** you have control over what gets changed
  - ▶ **collaborators** will be able to `push` to the repo

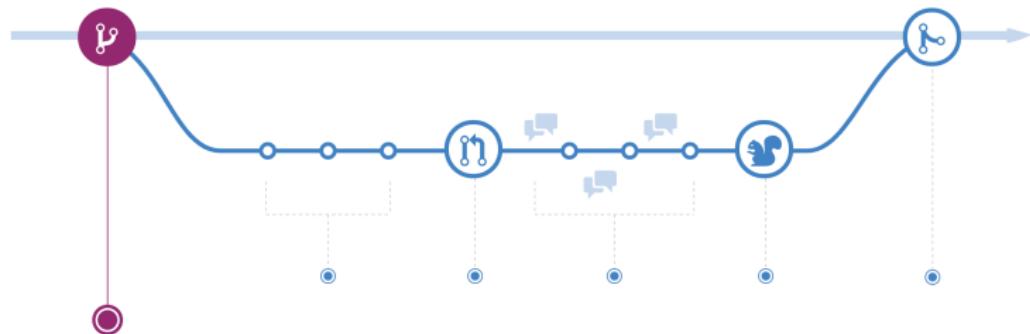
## a) **Collaborators:**

- ▶ work on a branch on the repo and create code
- ▶ send a `pull` request to add that code to the master repo

## b) **Owner:**

- ▶ comment on the `pull` request
- ▶ accept the `pull` request and/or `merge` the code

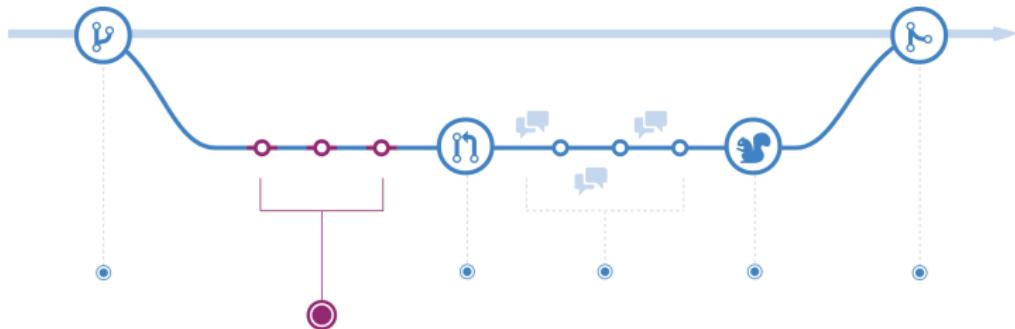
(1) a collaborator creates a branch to work on, that will eventually be merged back to the master branch



*Figure:* Understanding the GitHub flow

- ▶ changes in a branch do not affect the master branch
- ▶ ***ProTips***
  - ▶ anything in the master branch is deployment-ready
  - ▶ the branch should always be created off of the master branch

## (2) a collaborator works and commits on that branch



*Figure:* Understanding the GitHub flow

- ▶ use the same workflow in a branch: `git add`, `git status`, `git commit`
- ▶ ***ProTip***
  - ▶ use informative messages in your branch commits

### (3) a collaborator pushes to create a pull request



*Figure:* Understanding the GitHub flow

- ▶ a pull request notifies that your changes are ready to be reviewed and merged back to the master branch
- ▶ the review will validate that the changes do not create problems in the master branch and incorporate other members' comments

### (3) a collaborator pushes to create a pull request

The screenshot shows a GitHub repository page for 'marco-morales / testrepo'. The top navigation bar includes links for Code, Issues (0), Pull requests (1), Projects (1), Wiki, Insights, and Settings. A modal window titled 'Label issues and pull requests for new contributors' is open, explaining that GitHub will help potential first-time contributors discover issues labeled with 'help wanted' or 'good first issue'. The main content area displays a search bar with 'is:pr is:open' and filters for Labels and Milestones. Below this, a summary shows 1 Open and 1 Closed pull request, and a list for 'Melissabranch' with one item opened on Feb 7, 2018 by marco-morales. A ProTip at the bottom suggests using 'g 1' to go back to the issue listing page.

marco-morales / testrepo

Unwatch 2 Star 0 Fork 0

Code Issues 0 Pull requests 1 Projects 1 Wiki Insights Settings

Label issues and pull requests for new contributors

Now, GitHub will help potential first-time contributors discover issues labeled with [help wanted](#) or [good first issue](#)

Dismiss

Filters ▾ is:pr is:open Labels Milestones New pull request

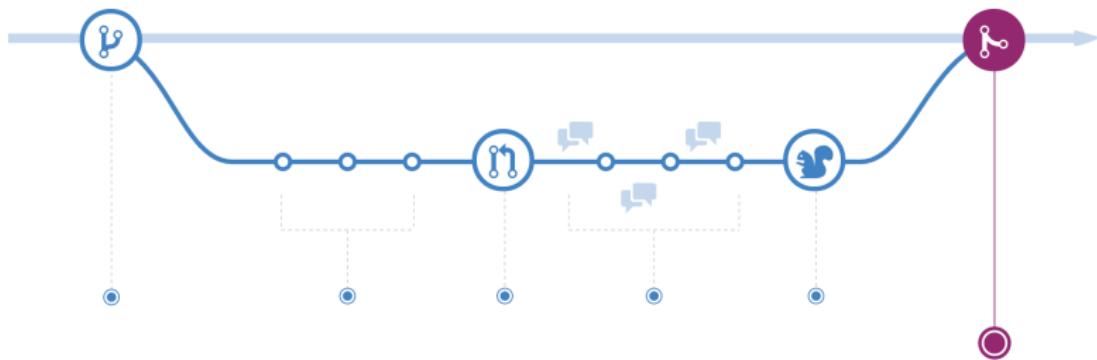
1 Open ✓ 1 Closed

Author ▾ Projects ▾ Labels ▾ Milestones ▾ Reviews ▾ Assignee ▾ Sort ▾

Melissabranch  
#2 opened on Feb 7, 2018 by marco-morales

ProTip! Type `g 1` on any issue or pull request to go back to the issue listing page.

## (4) an owner reviews changes, resolves conflicts, and approves the merge



*Figure:* Understanding the GitHub flow

- ▶ once the proposed changes have been validated they are merged back into the `master` branch
- ▶ the merge preserves records of changes made on the branch

## (4) an owner reviews changes, resolves conflicts, and approves the merge

The screenshot shows a GitHub pull request interface for a repository named "marco-morales / testrepo". The pull request is titled "Melissabranch #2" and is marked as "Open". It shows a diff between the "master" branch and the "melissabranch" branch, specifically focusing on a file named "testfile.R". The diff highlights changes made by the author:

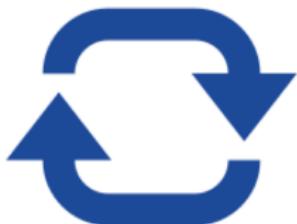
```
@@ -3,4 +3,4 @@
 3     ## this is a test fi
 4
 5     print("Hello World!")
 6 -   print(":")
 6 +   print("Hello World!!")
```

The "Comment" section of the review interface is open, showing three options:

- Comment: Submit general feedback without explicit approval.
- Approve: Submit feedback and approve merging these changes.
- Request changes: Submit feedback that must be addressed before merging.

At the bottom of the review interface is a green "Submit review" button.

rinse and repeat



# a quick exercise

# a quick exercise

from the command line:

- ▶ go to a brand new location
- ▶ clone somebody else's remote repo

```
$ git clone  
https://github.com/marco-morales/testrepo.git
```

- ▶ (checkout and) create a branch

```
$ git checkout -b mytestbranch-myname
```

- ▶ make a change in your code file
- ▶ go on, verify that git is tracking the change

```
$ git status
```

# a quick exercise

from the command line:

- ▶ do your usual git routine

```
$ git add testfile.R
```

```
$ git commit -m "Add hubris to the code"
```

- ▶ now, you'll create a pull request

```
$ git push origin mytestbranch-myname
```

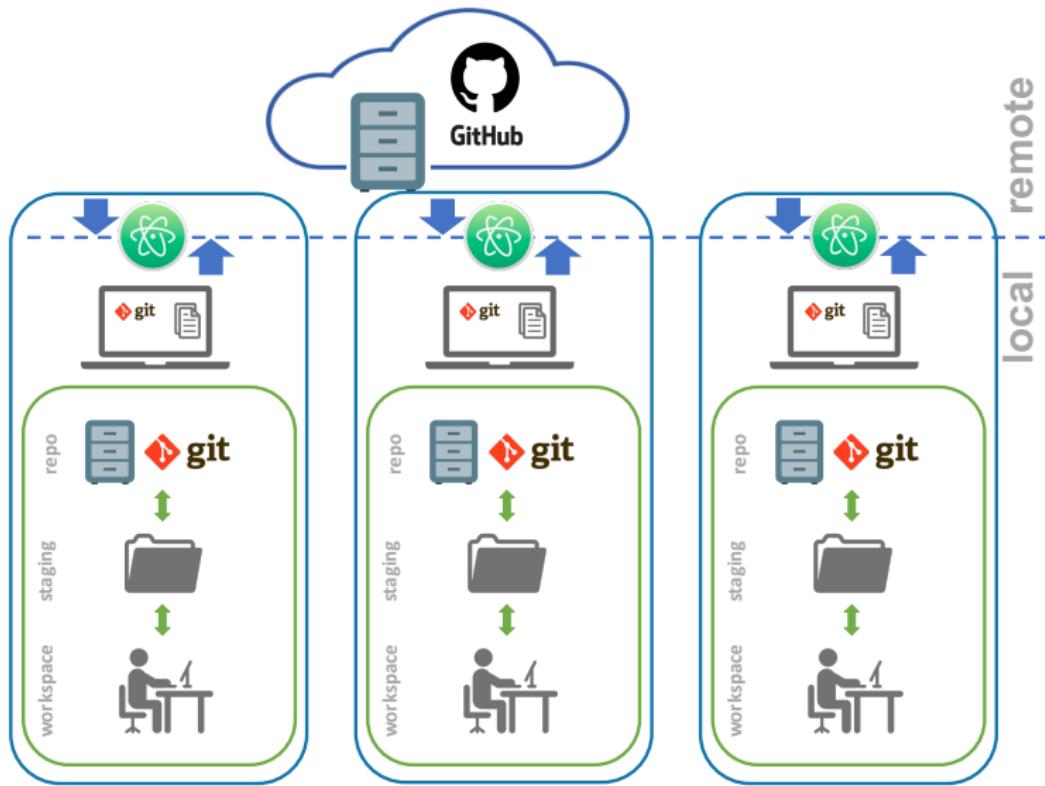
- ▶ time for the repo owner to intervene!

# Your Friendly Neighborhood Atom

# a simpler workflow is possible

- ▶ we've been working with git+GitHub from the **command line**
  - ▶ it can get confusing and harder than necessary at times
- ▶ now that you understand the flow, we can make our lives easier with an **IDE** (Integrated Development Environment)
- ▶ my preferred IDE today → **Atom**
  - ▶ it can seamlessly handle git + GitHub interaction, and coding needs

# a git+GitHub workflow through an IDE



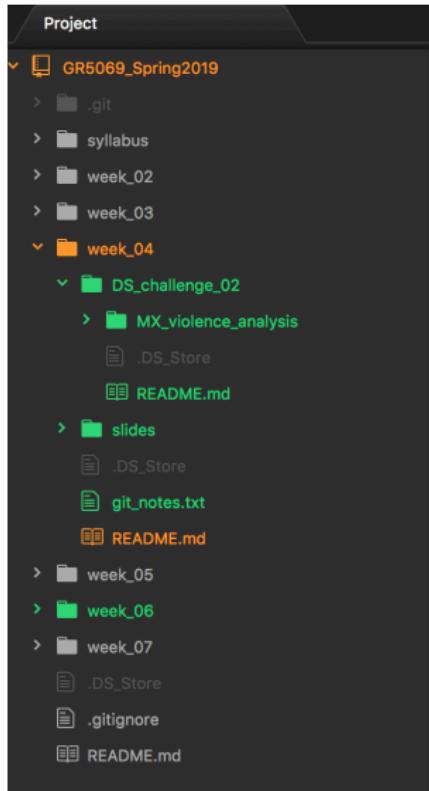
# a git+GitHub workflow through an IDE

The screenshot shows a GitHub interface integrated into an IDE. On the left, the project structure for 'git009\_Spring2019' is visible, including files like README.md, week\_01, week\_02, week\_03, week\_04, week\_05, week\_06, week\_07, week\_08, and week\_09. The main pane displays the 'README.md' file with the following content:

```
21 21  the GitHub YouTube channel [GitHub training series] (https://www.youtube.com/GitHub)
22 22 * [GitHub Learning Lab](https://lab.github.com) hands-on tutorials put together by GitHub
23 23 * [Understanding the GitHub flow](https://guides.github.com/introduction/flow/) is a 5-minute tutorial
24 24 to understand the basics of team collaboration through GitHub
25 25 * [GitHub Help](https://help.github.com/) is a great place to start at the source
26 26
27 27 ## Atom
28 28 Although it's probably best to start with the command line, once you're comfortable coding in a text
29 29 editor, it might start to make sense to also interact with git and GitHub from your text editor as
30 30 well. Atom is an open source text editor - supported by GitHub - with a nice integration with git and
31 31 GitHub.
32 32 * [Version Control in Atom](https://flight-manual.atom.io/using-atom/sections/version-control-in-
33 33 atom/) and [GitHub package](https://flight-manual.atom.io/using-atom/sections/github-package/) chapters
34 34 from the [Atom Flight Manual](https://flight-manual.atom.io) to understand the basics of working with
35 35 git and GitHub from Atom
36 36 * [Atom Flight Manual](https://flight-manual.atom.io) if you want to learn more about other packages
37 37 * that will make it easier to code in your language of choice
```

The right side of the interface shows the GitHub status bar with 'Unstaged Changes: week\_04/README.md'. The GitHub sidebar includes sections for 'Unstaged Changes' (with files like week\_04/week\_04\_challenge\_02/MX\_violence\_analysis/r/functions.R, week\_04/week\_04\_challenge\_02/README.md, etc.) and 'Staged Changes' (empty). A commit message field is present at the bottom.

# a simple way to identify the git status of each file

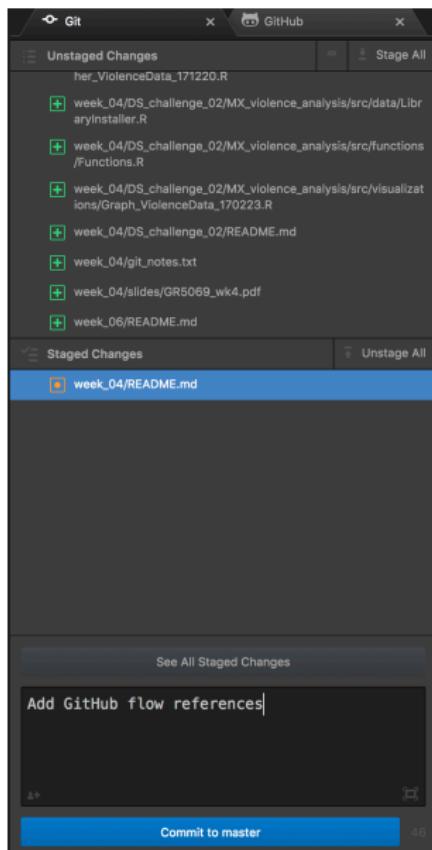


# a simple way to git diff on a file

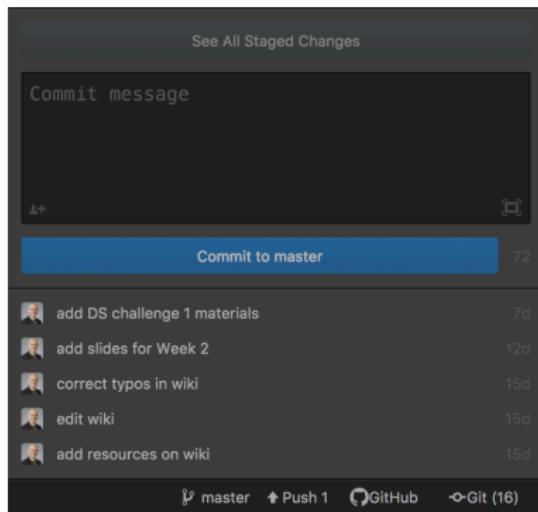
The screenshot shows a GitHub interface with the tab 'Unstaged Changes: week\_04...' selected. The main area displays the content of README.md with several changes highlighted in red and green, indicating additions and deletions. The changes are as follows:

```
@@ -21,11 +21,12 @@ Git and GitHub have done a great job of putting together and facilita...
21 21
22 * the GitHub YouTube channel [GitHub training series](https://www.youtube.com/GitHub)
23 * [GitHub Learning Lab](https://lab.github.com) hands-on tutorials put together by GitHub
24 * [Understanding the GitHub flow](https://guides.github.com/introduction/flow/) is a 5-minute tutorial
  • to understand the basics of team collaboration through GitHub
25 * [GitHub Help](https://help.github.com/) is a great place to start at the source
26
27 ## Atom
28 Although it's probably best to start with the command line, once you're comfortable coding in a text
  • editor, it might start to make sense to also interact with git and GitHub from your text editor as
  • well. Atom is an open source text editor – supported by GitHub – with a nice integration with git and
    GitHub.
29 Although it's probably best to start with the command line, once you're comfortable coding in a text
  • editor, it might start to make sense to also interact with git and GitHub from your text editor as
  • well. Atom is an open source text editor – supported by GitHub – with a nice integration with git and
    GitHub (aka an IDE).
30 30
31 * the [Version Control in Atom](https://flight-manual.atom.io/using-atom/sections/version-control-in-
  • atom/) and [GitHub package](http://flight-manual.atom.io/using-atom/sections/github-package/) chapters
  • from the [Atom Flight Manual](https://flight-manual.atom.io) to understand the basics of working with
  • git and GitHub from Atom
32 * [Atom Flight Manual](https://flight-manual.atom.io) if you want to learn more about other packages
  • that will make it easier to code in your language of choice
```

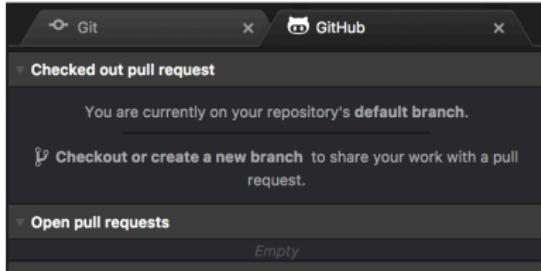
all your git workflow (add, commit) on a single screen



# all your git workflow (push) on a single screen



# all your GitHub workflow on a single screen



**Though this be madness,  
yet there's method in't**

## Recap: the method to this version control madness...

- ▶ basic **actions** to master in git
  - ▶ `git init`: initializes git, and indicates that the folder should be tracked
  - ▶ `git add`: brings new files to the attention of git to be tracked as well
  - ▶ `git commit`: takes a snapshot of alerted files
  - ▶ `git push`: sends changes committed in your branch (of your local repo) to the remote branch (of the GitHub repo)

# Recap: the method to this version control madness...

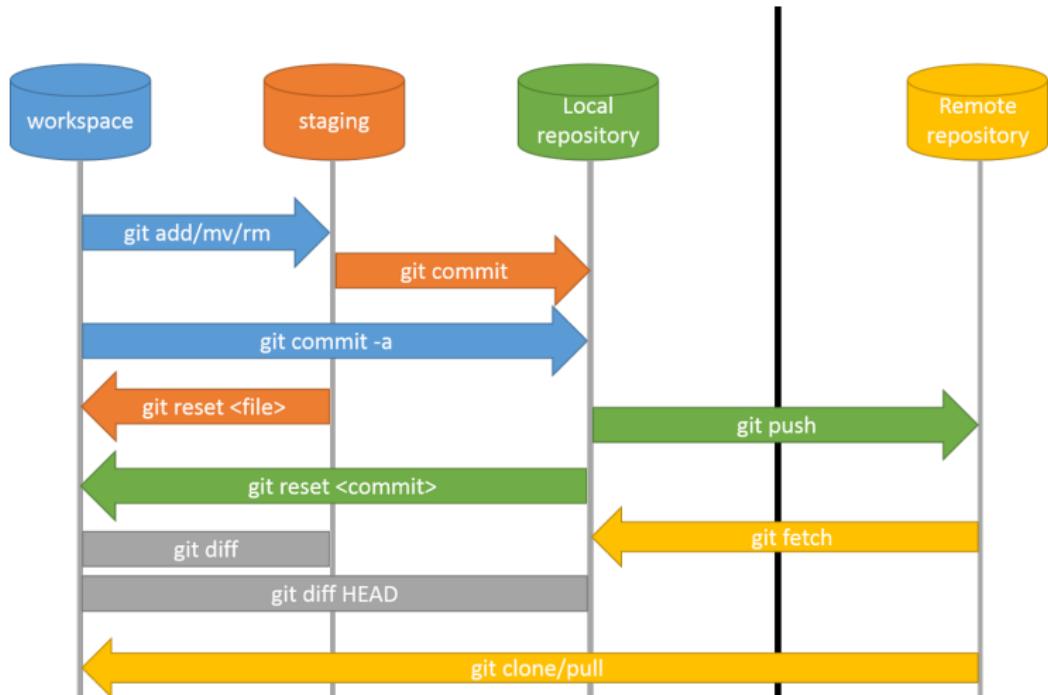


Figure: Pro Git, 2nd Edition

# Version Control

Marco Morales  
[marco.morales@columbia.edu](mailto:marco.morales@columbia.edu)

GR5069  
Topics in Applied Data Science  
for Social Scientists

Spring 2019  
Columbia University