Version Control

Marco Morales
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**

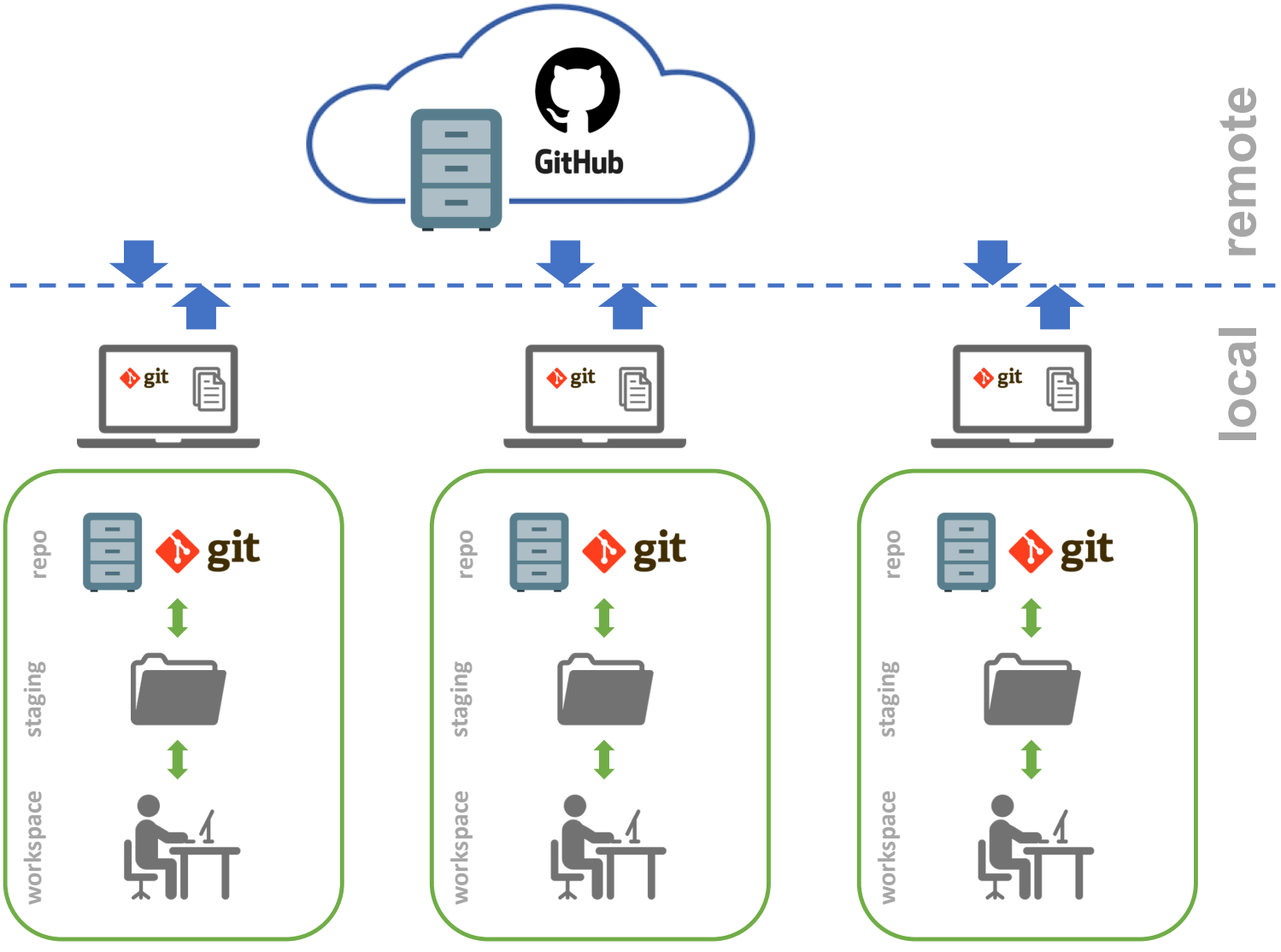
- your project should also work for **larger data sets** and/or be on the path of **automation**

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

RECAP: Version control

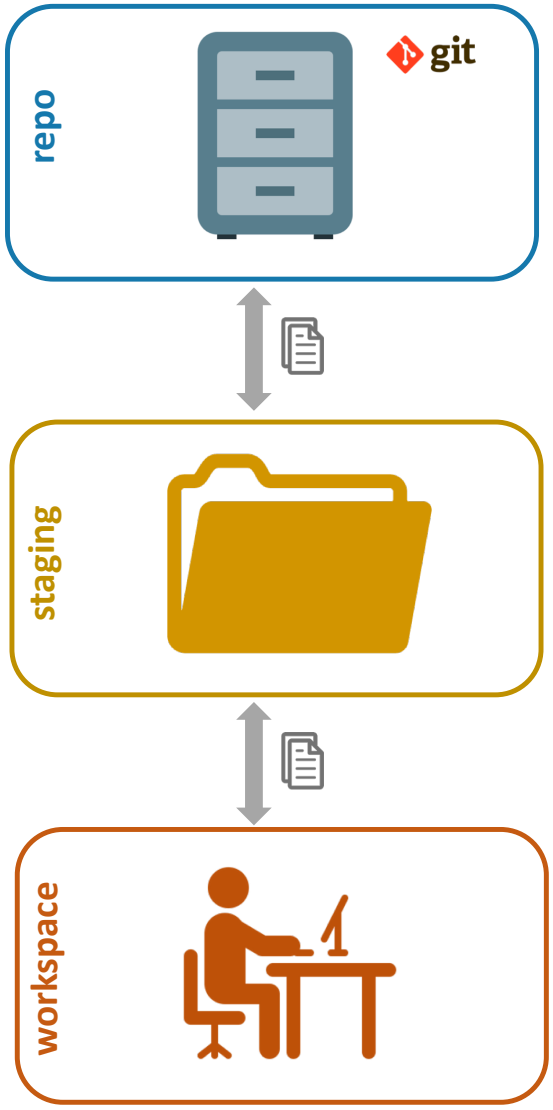
- **version control** allows you to keep track of changes / progress in your code
  - keeps "**snapshots**" of your code over time
  - helpful to **debug**, and to enhance **reproducibility**
  - also great for **team collaboration** (everyone can see who changed what!)
- **Git** is a version control 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"
```

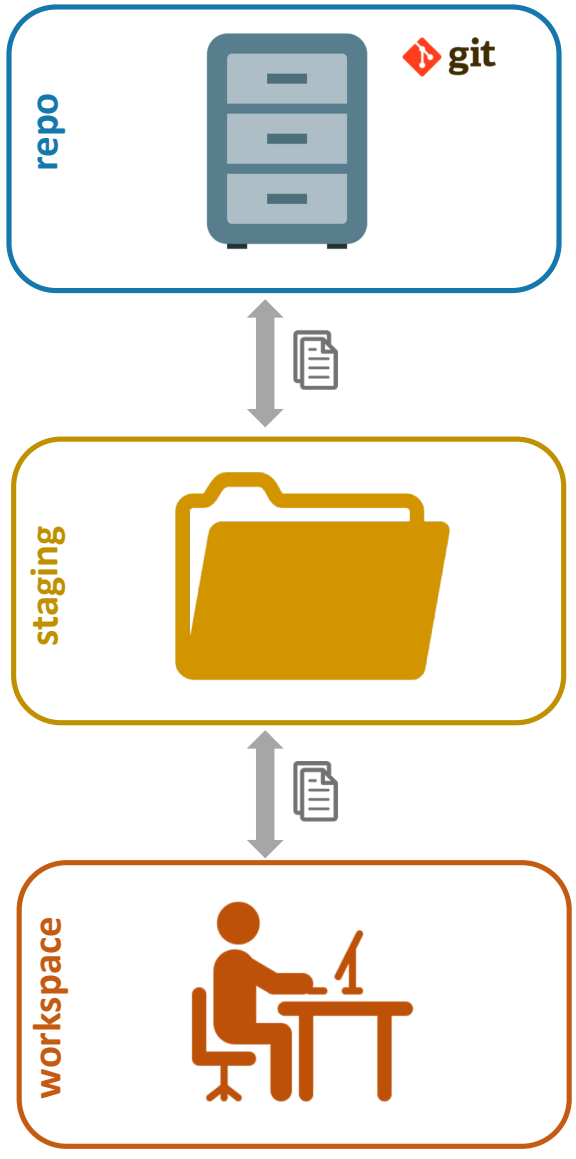
```
$ git config --global user.email johndoe@example.com
```

- **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
```

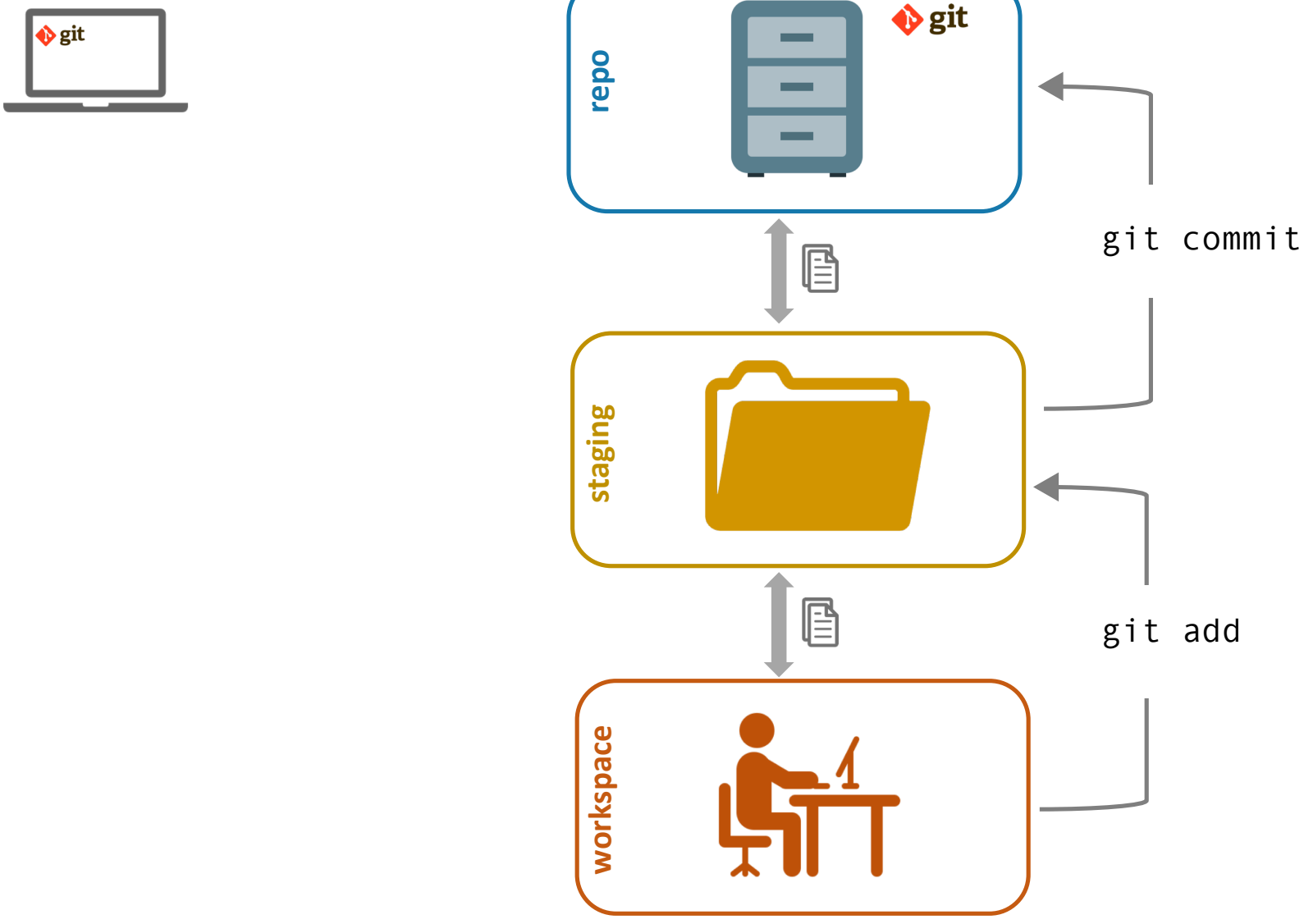
- add files you want git to ignore in the `.gitignore` file

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

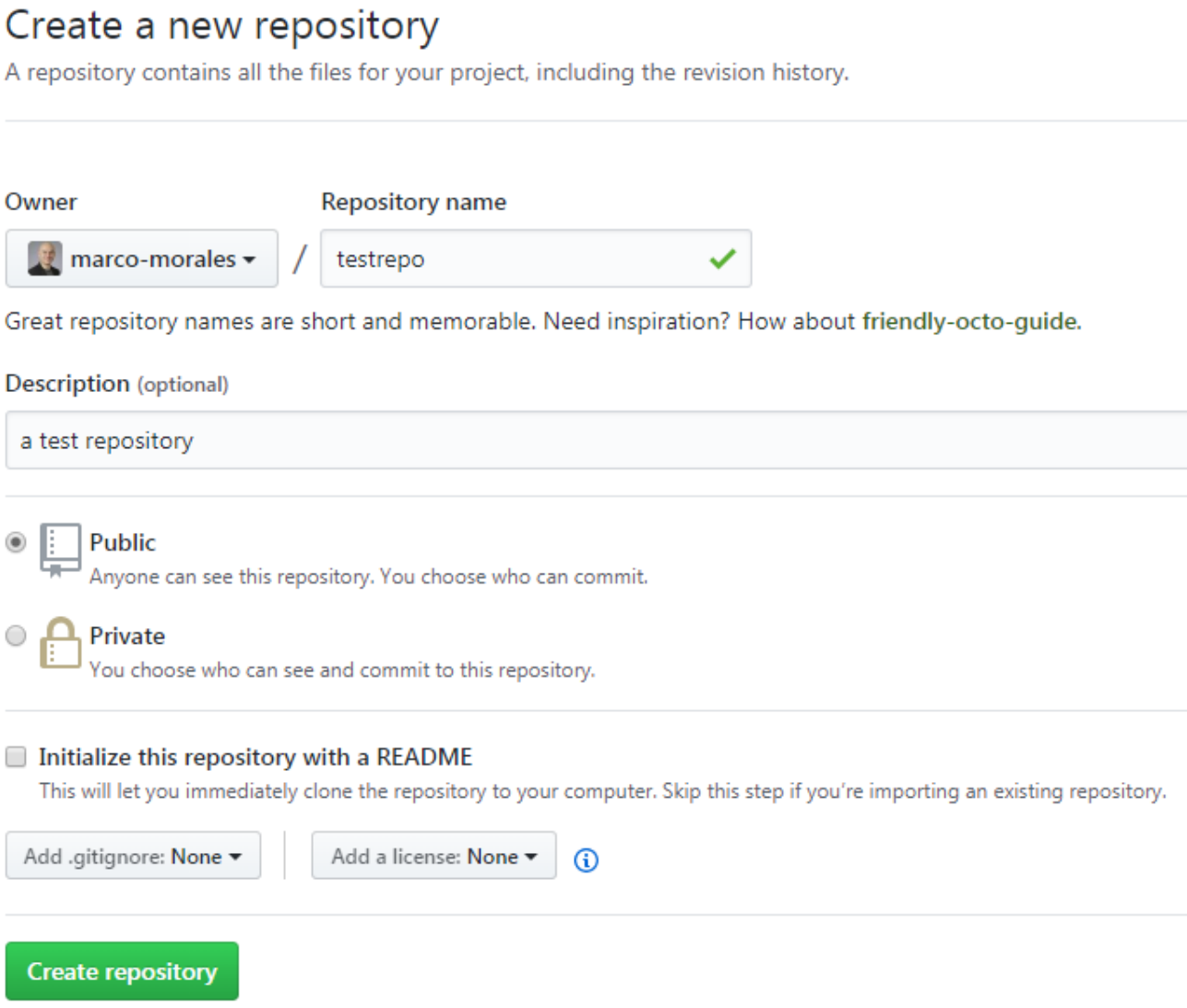
- a file will be committed **exactly** as it was when you `git add`-ed it
- if you change the file **after** you `git add` it, and want to commit the new changes, you need to `git add` again before the `git commit`
- use `git status` to assess what's being staged and will be commited

git workflow **ProTips**

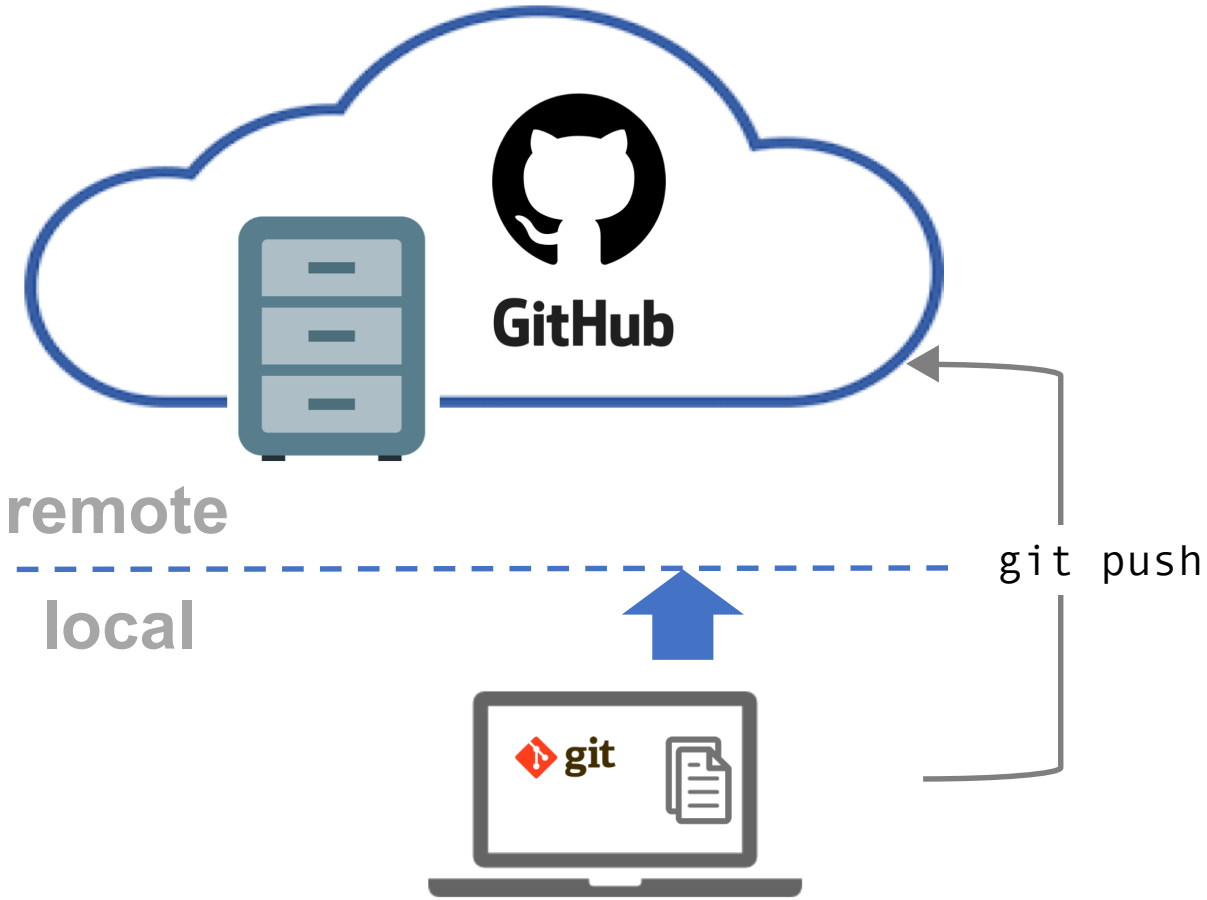
- **NEVER** use `git add .`
- use `git status` often as **validation**
- only `add` and `commit` **source files**
  - omit files you can reproduce using source files
- `commit` **small chunks of logically grouped changes**
  - you may want to undo a change, and only that change
- `commit` with **informative**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

push globally
(to GitHub)

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



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”)

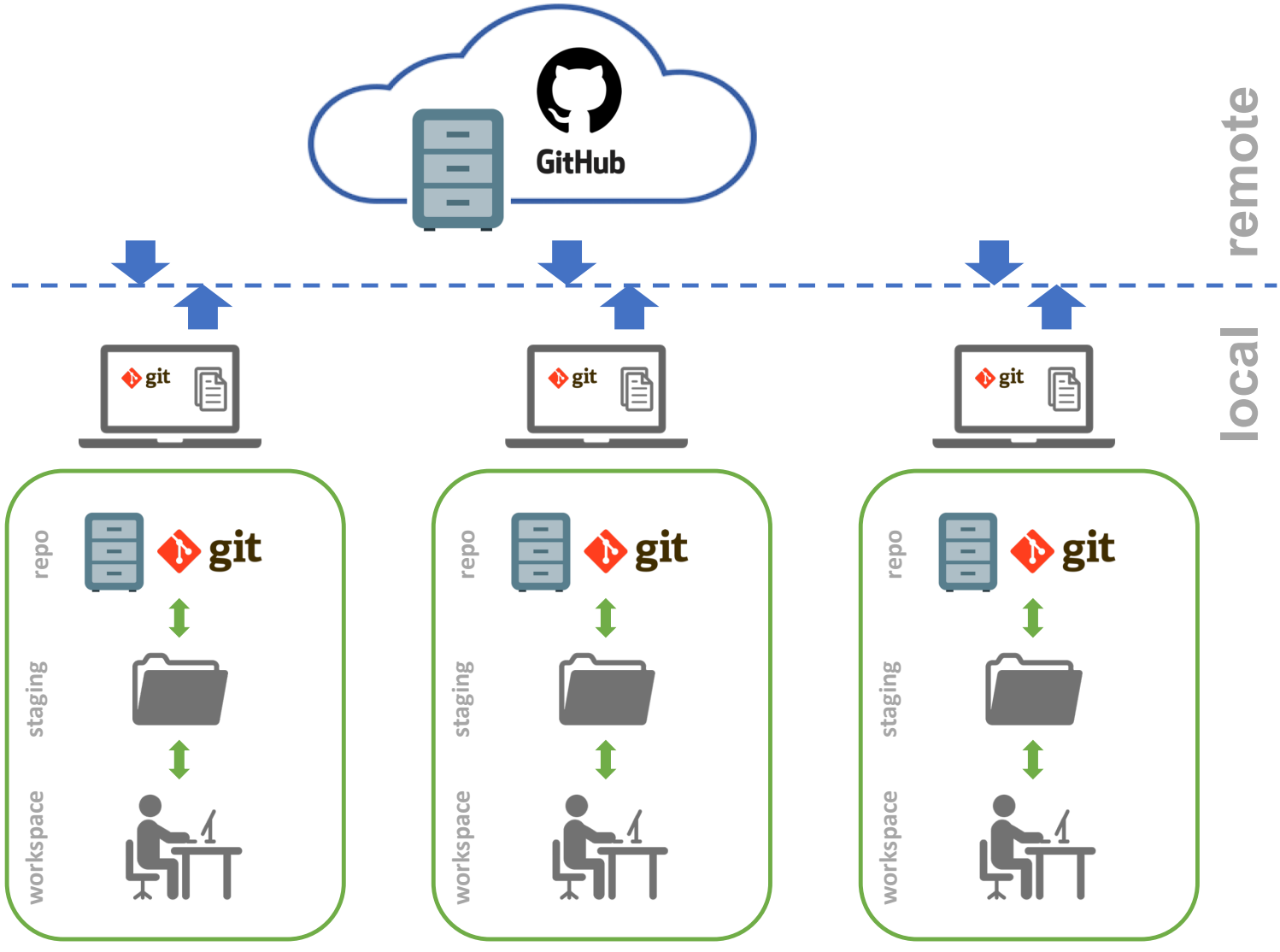
```
$ git remote add origin
https://github.com/marco-morales/testrepo.git
```

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

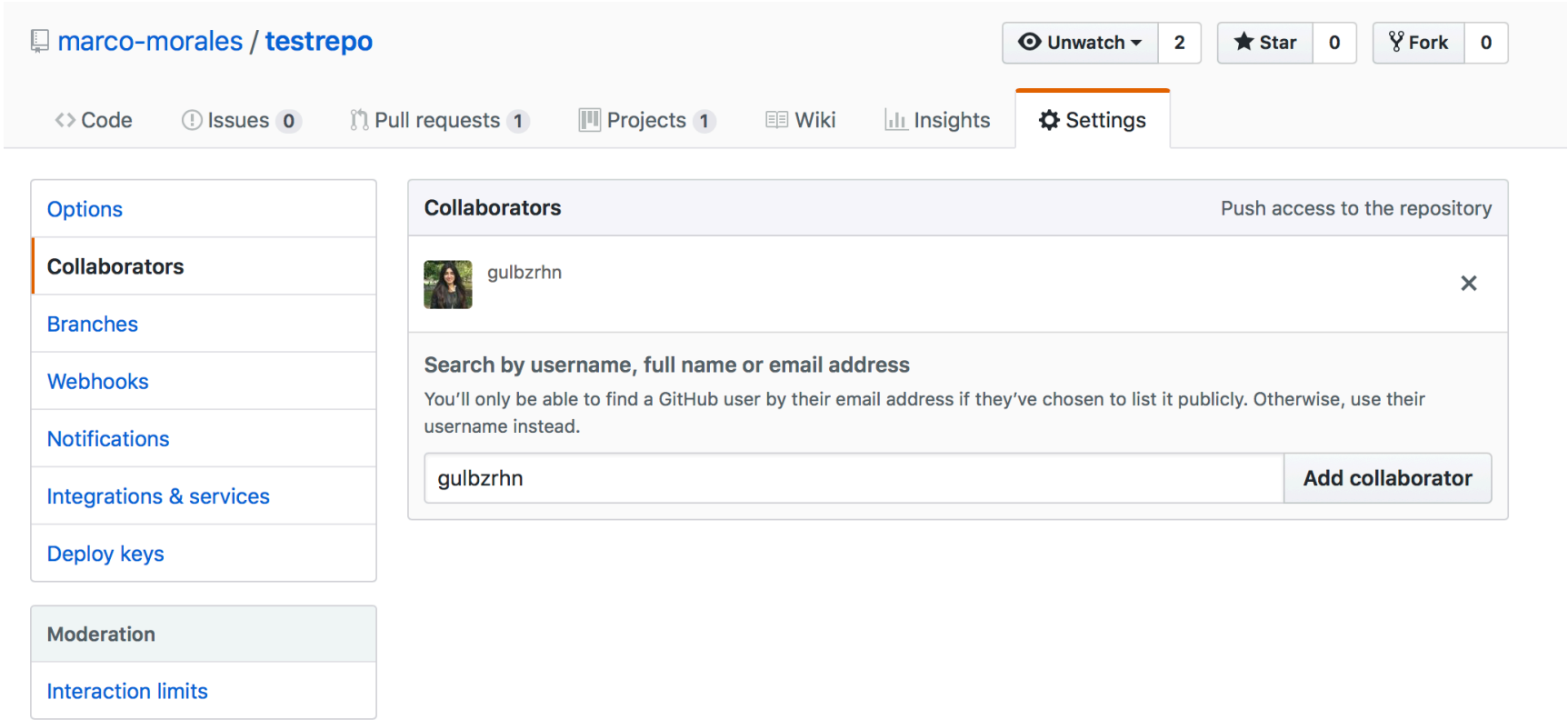
```
$ git push -u origin master
```

git+GitHub for team collaboration

all the building blocks are now in place



now, enable collaborators in your GitHub repo



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

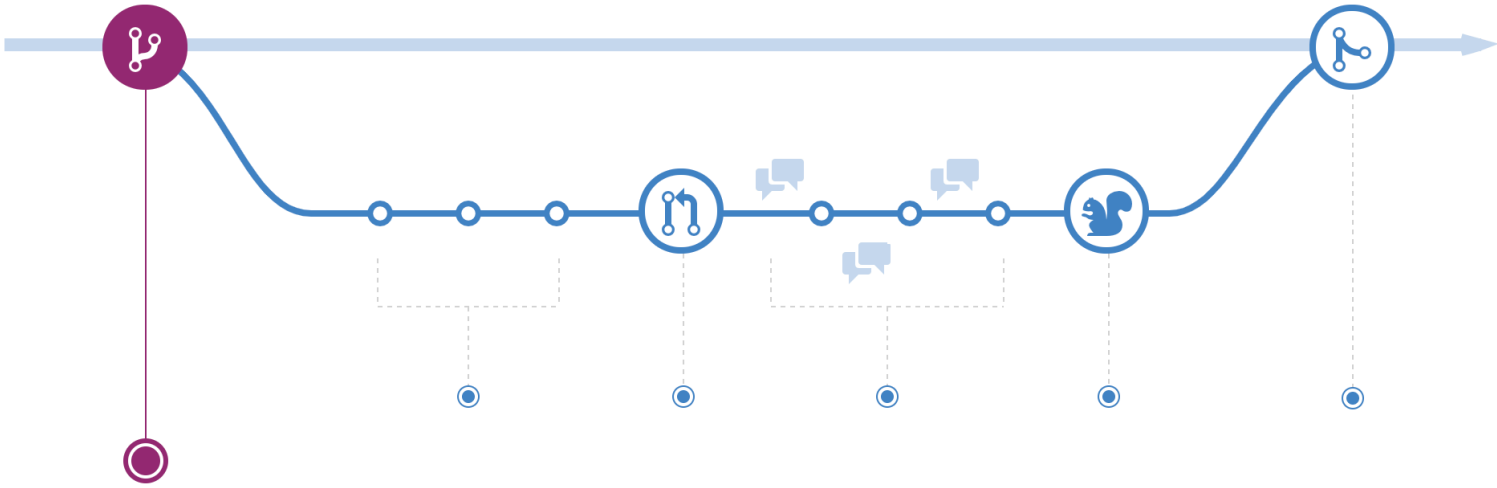


Figure: Understanding the GitHub flow

- 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 `commit`s on that branch

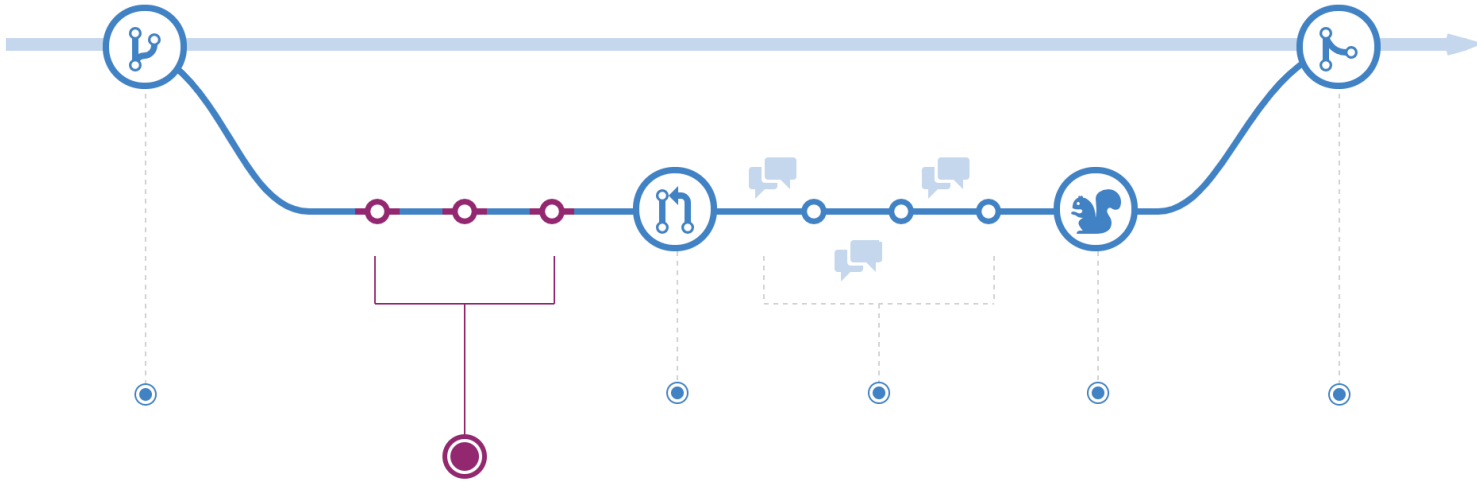


Figure: Understanding the GitHub flow

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

(3) a collaborator `push`es to create a pull request

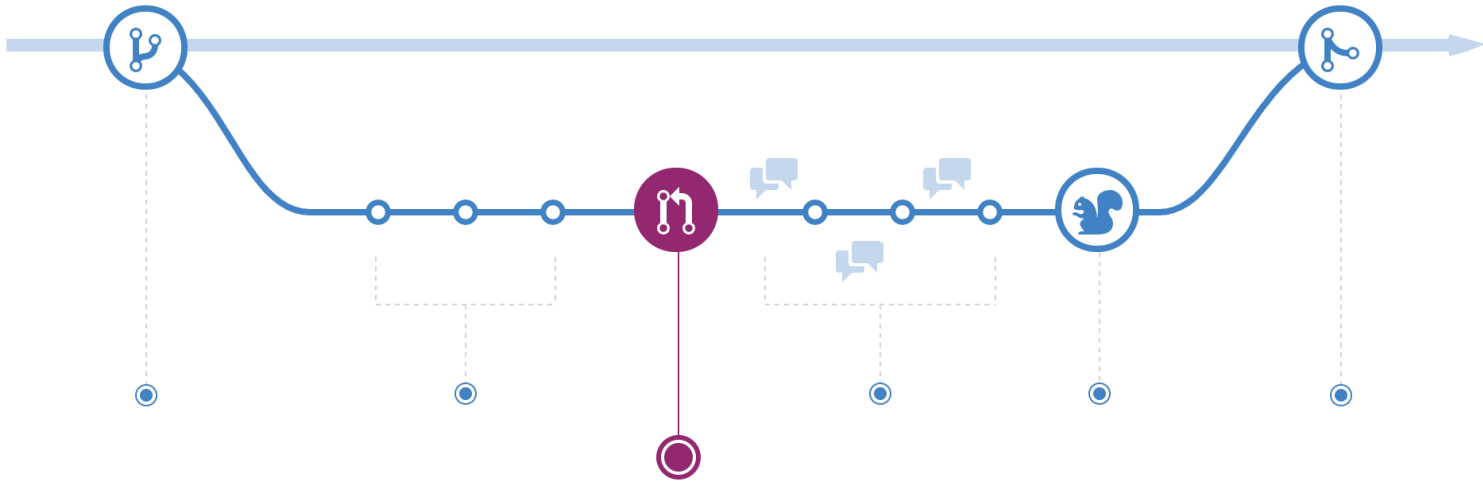
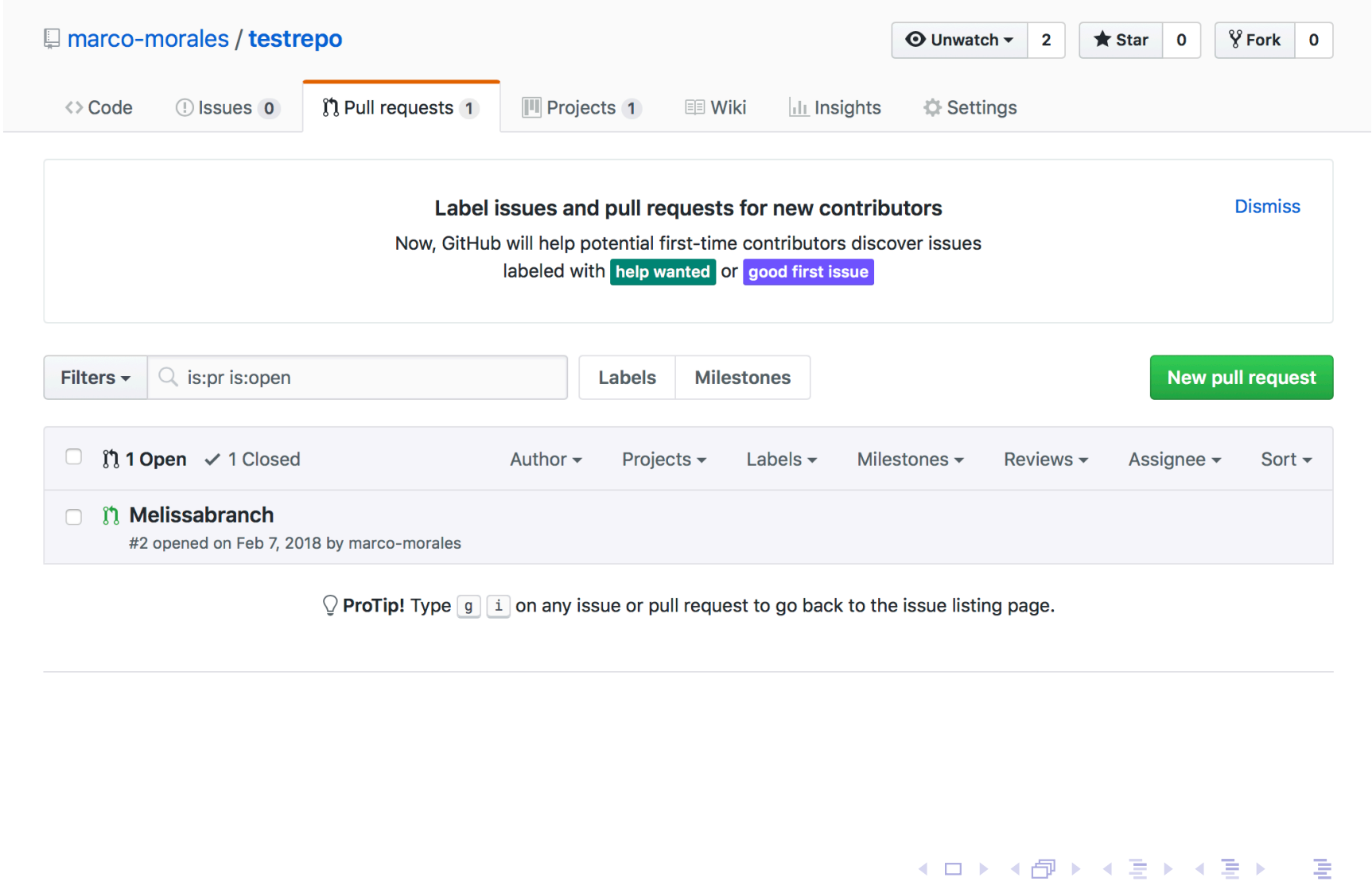


Figure: Understanding the GitHub flow

- 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 incoporate other members' comments

(3) a collaborator `push`es to create a pull request



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

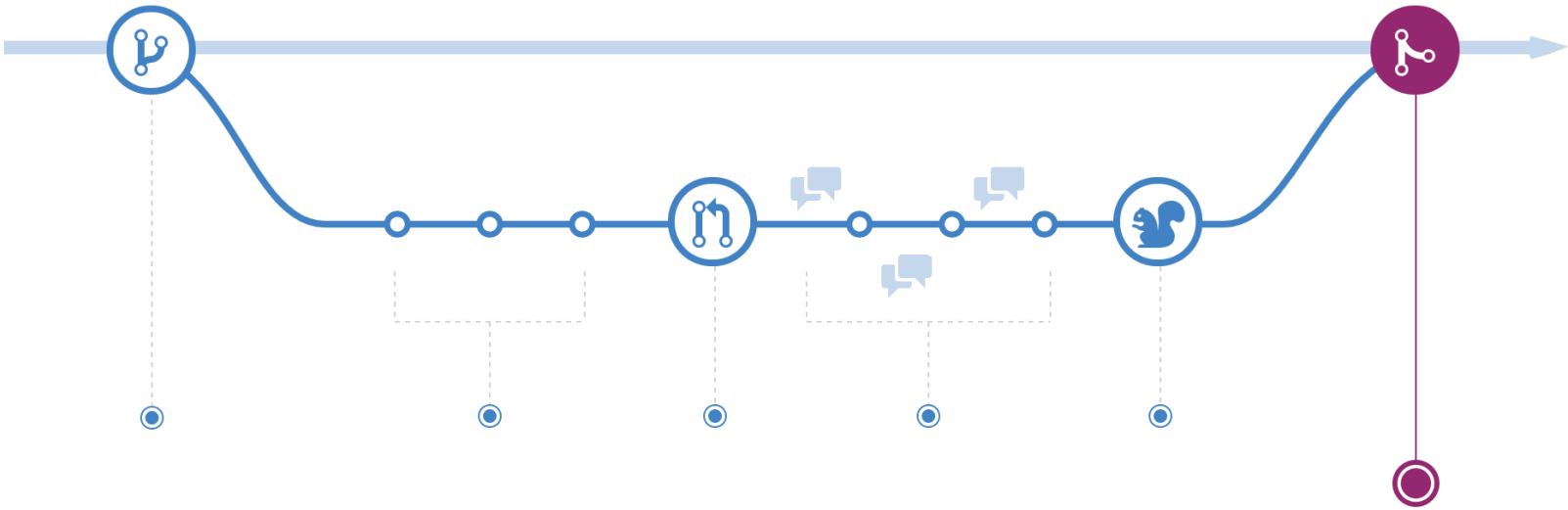
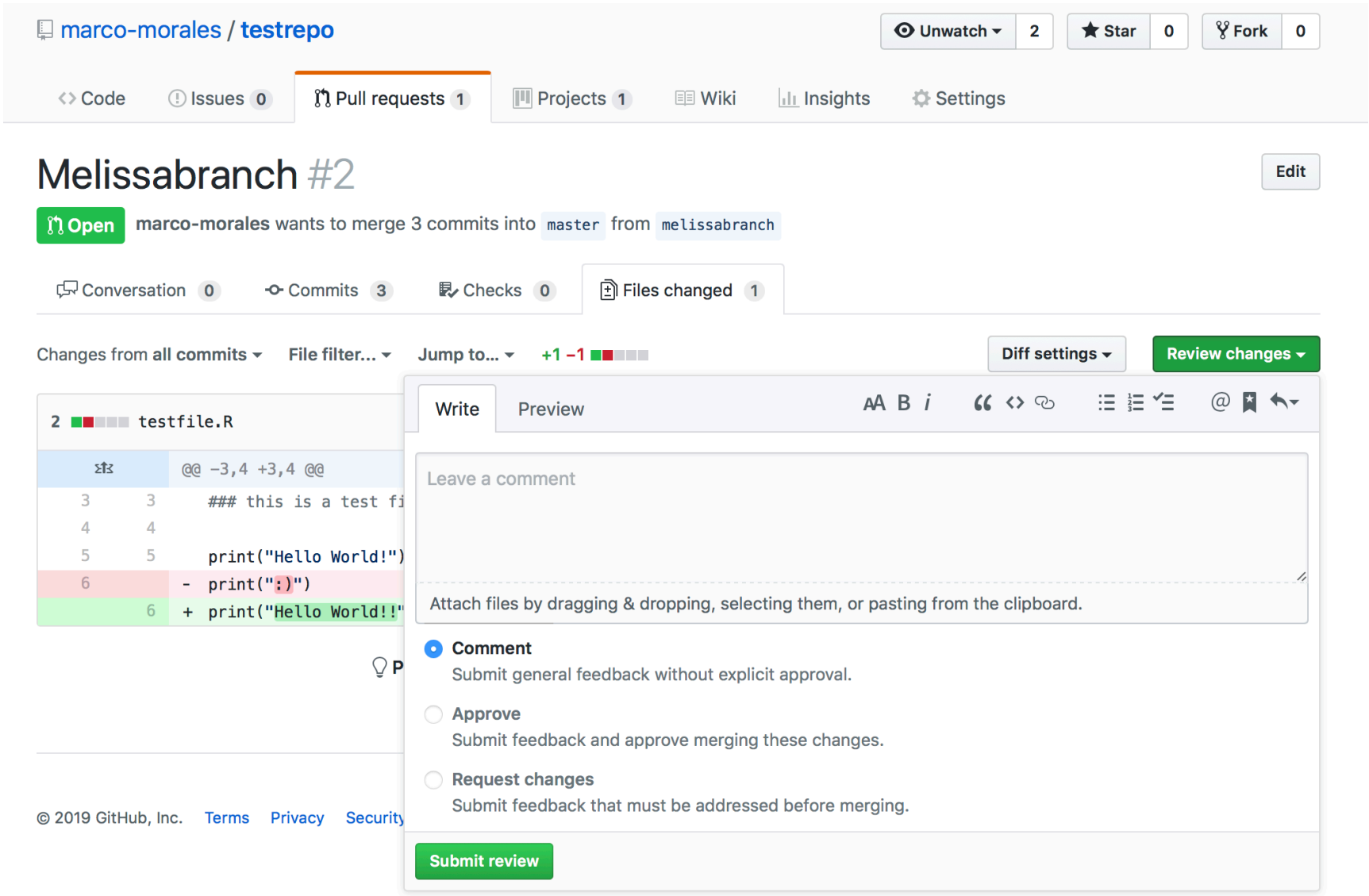


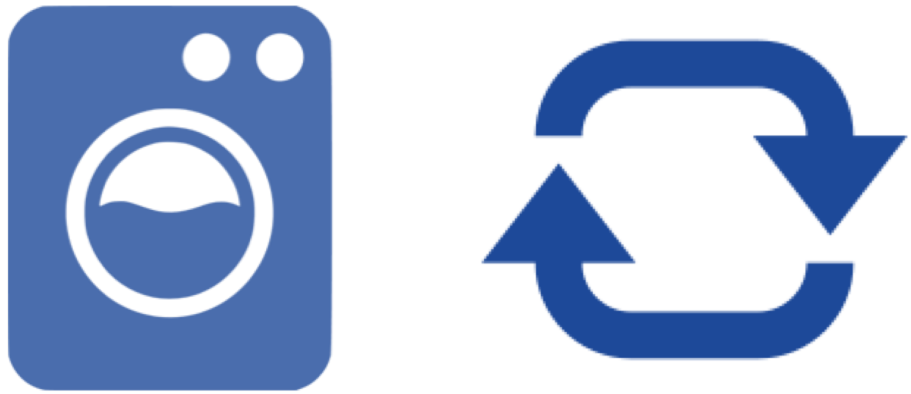
Figure: Understanding the GitHub flow

- 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



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
```

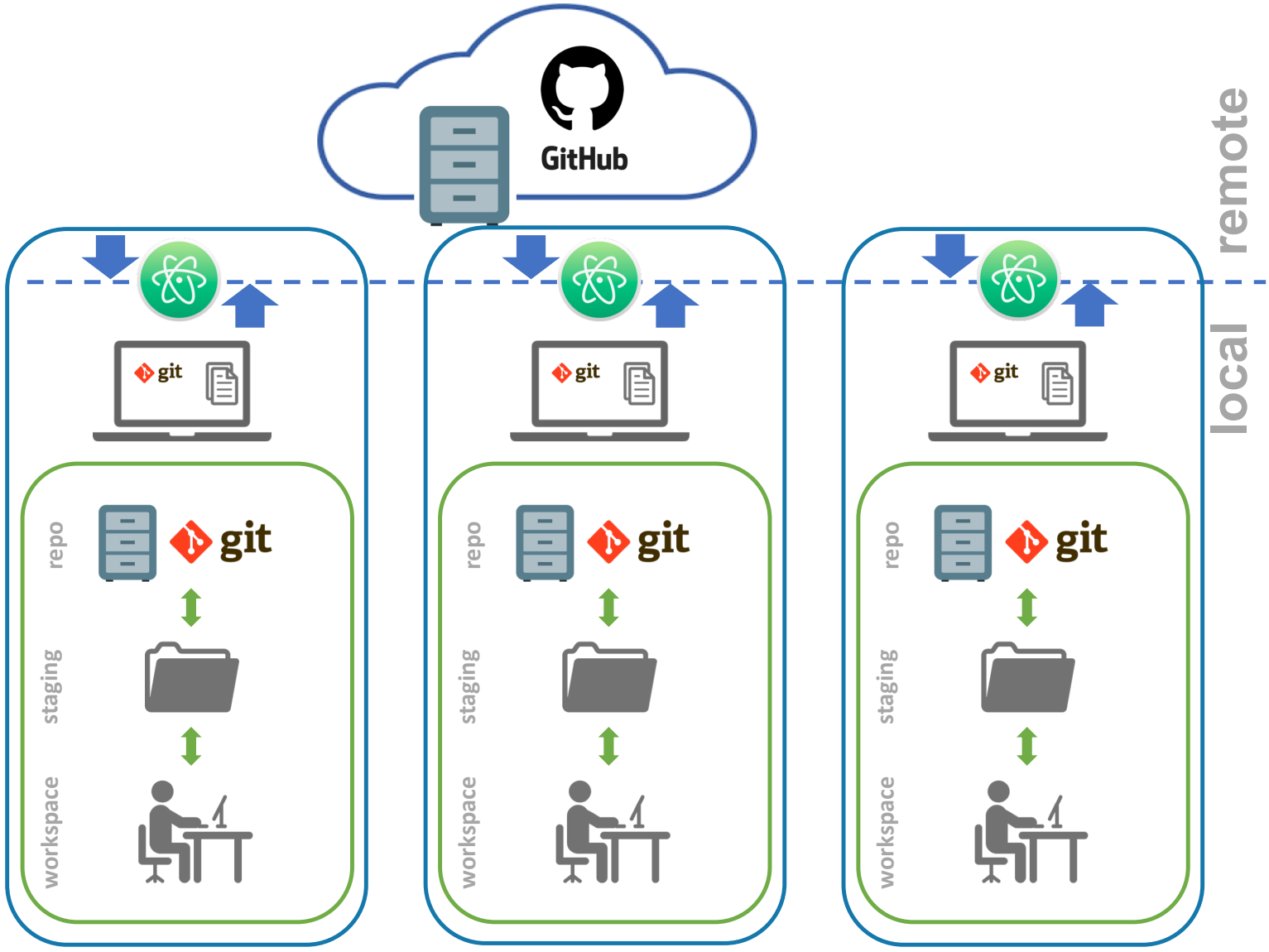
- time for the repo owner to intervene!

Your Friendly Neighborhood Atom

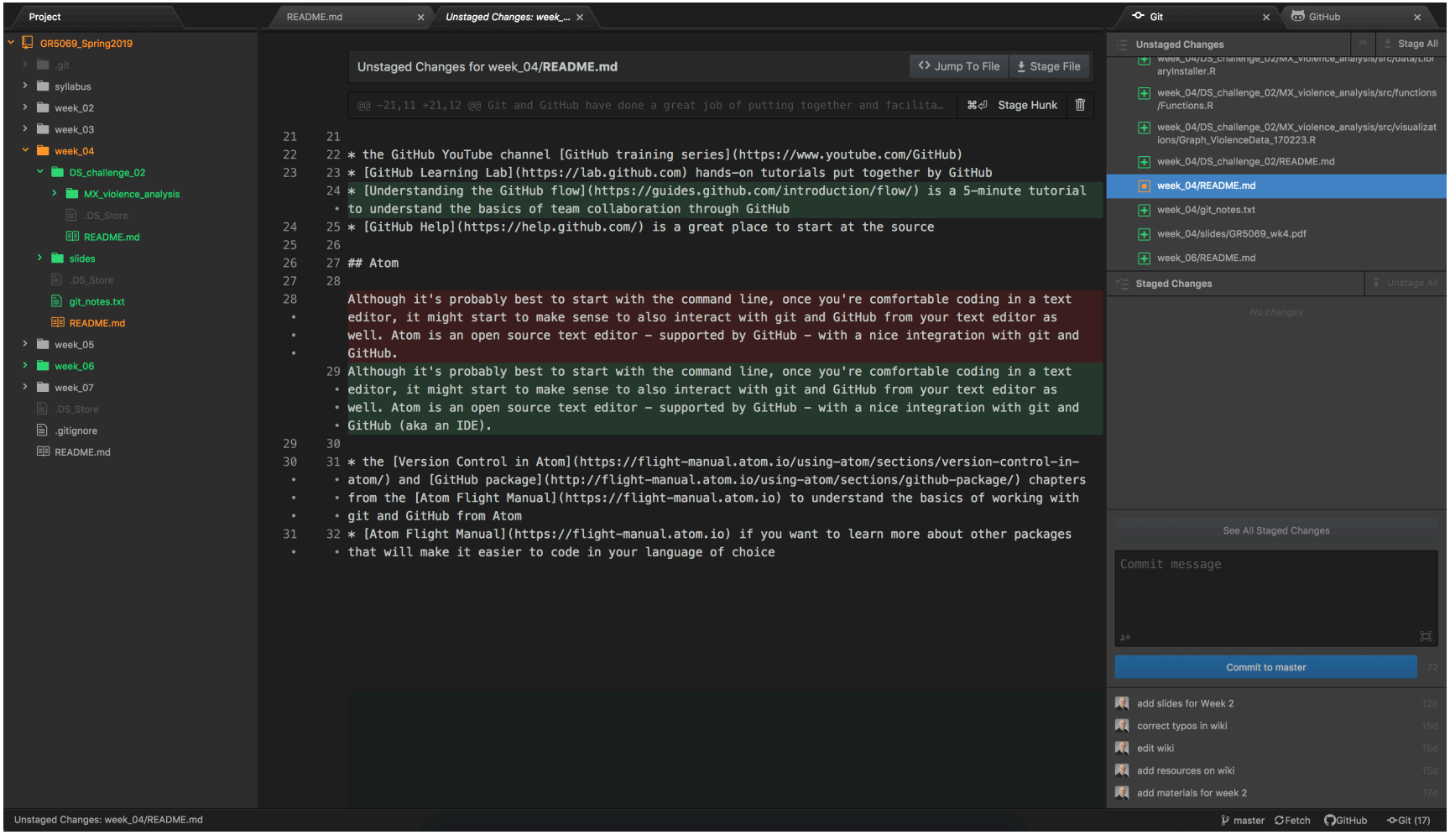
a simpler workflow is possible

- we've been working with git+GitHub from the **command line**
  - it can get confusing and harder that 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 $\rightarrow$ **Atom**
  - it can seamlessly handle git + GitHub interaction, and coding needs

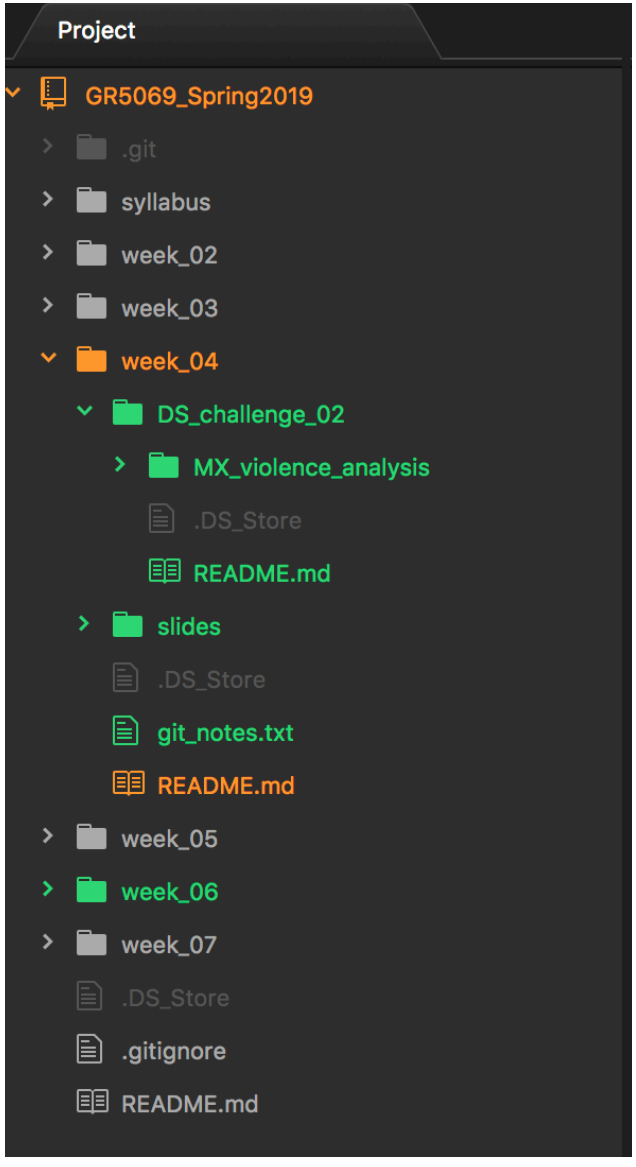
a git+GitHub workflow through an IDE



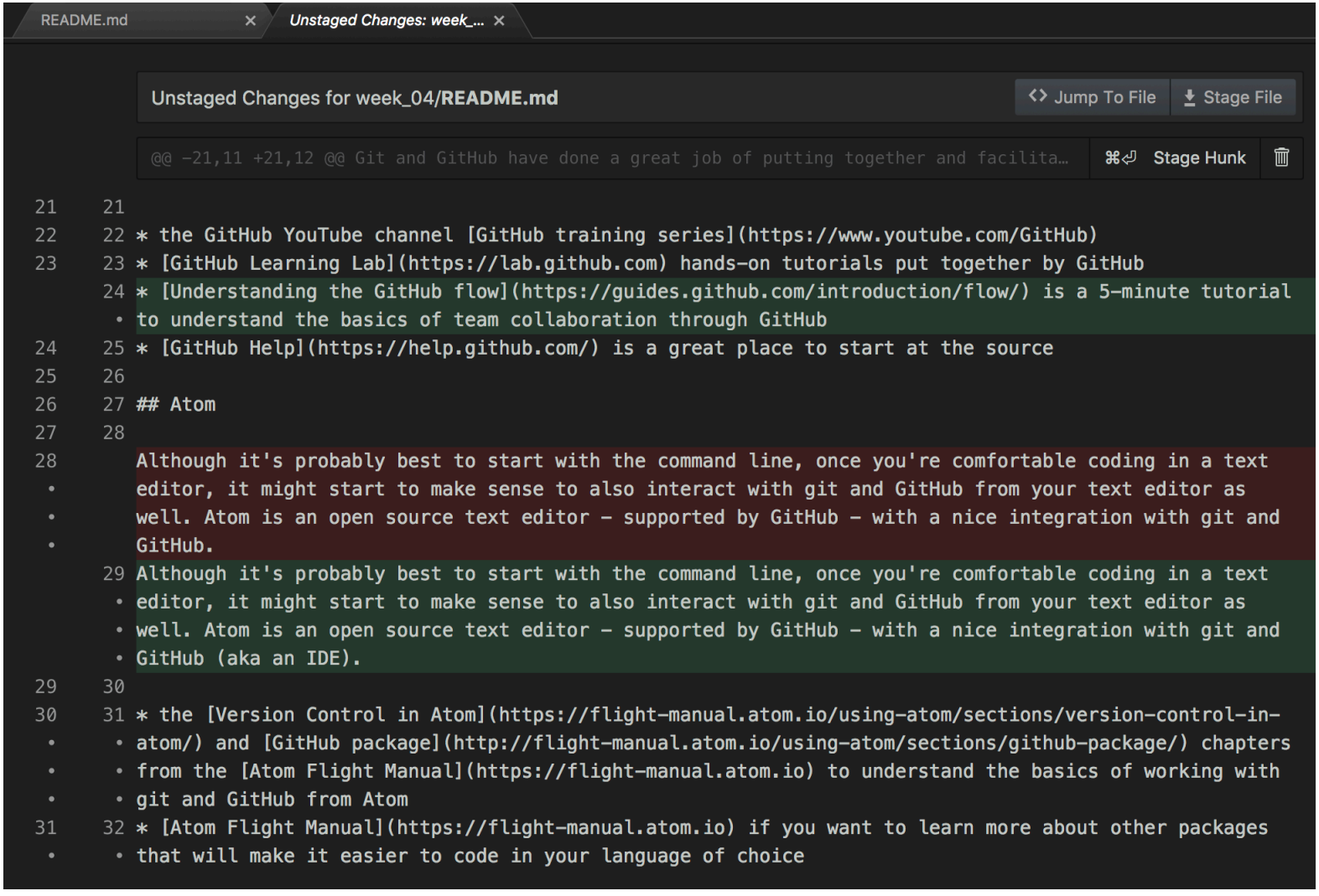
a git+GitHub workflow through an IDE



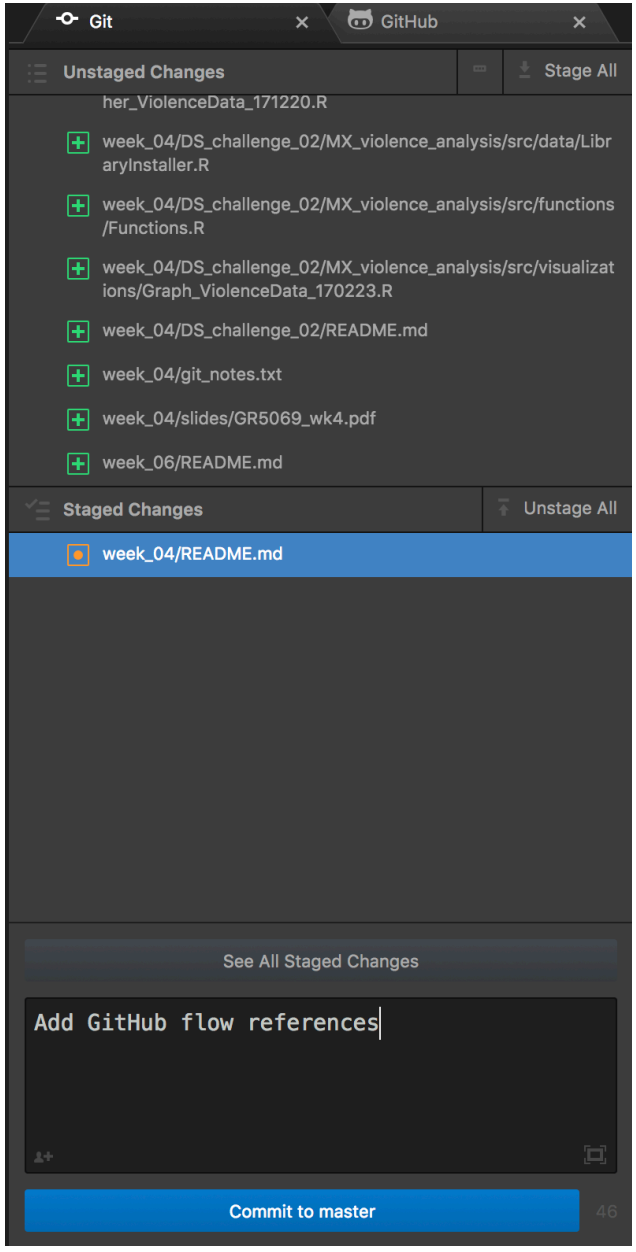
a simple way to identify the `git status` of each file



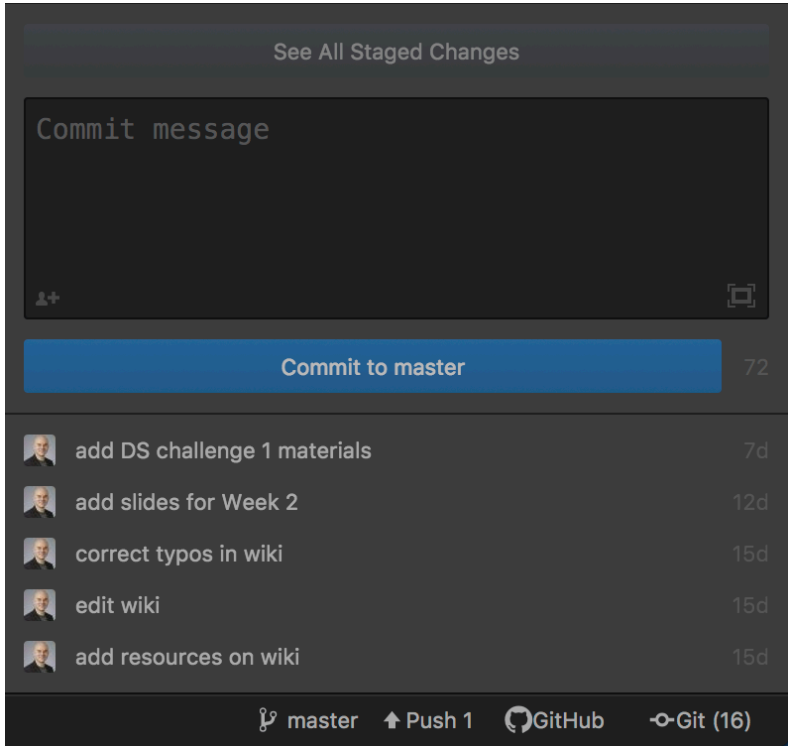
a simple way to `git diff` on a file



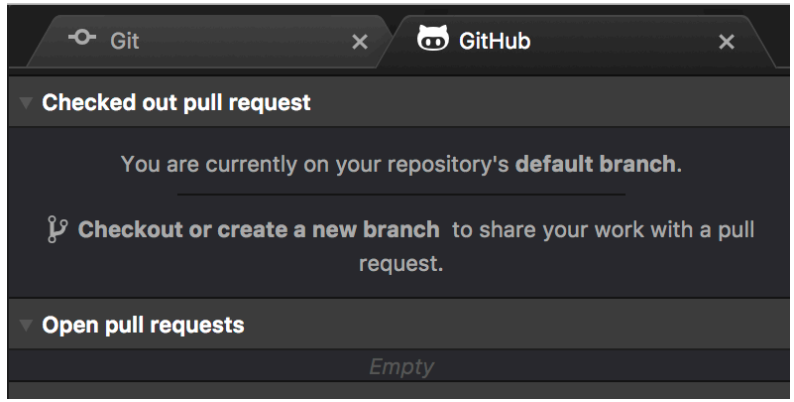
all your git workflow (`add`, `commit`) on a single screen



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 commited in your branch (of your local repo) to the remote branch (of the GitHub repo)

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

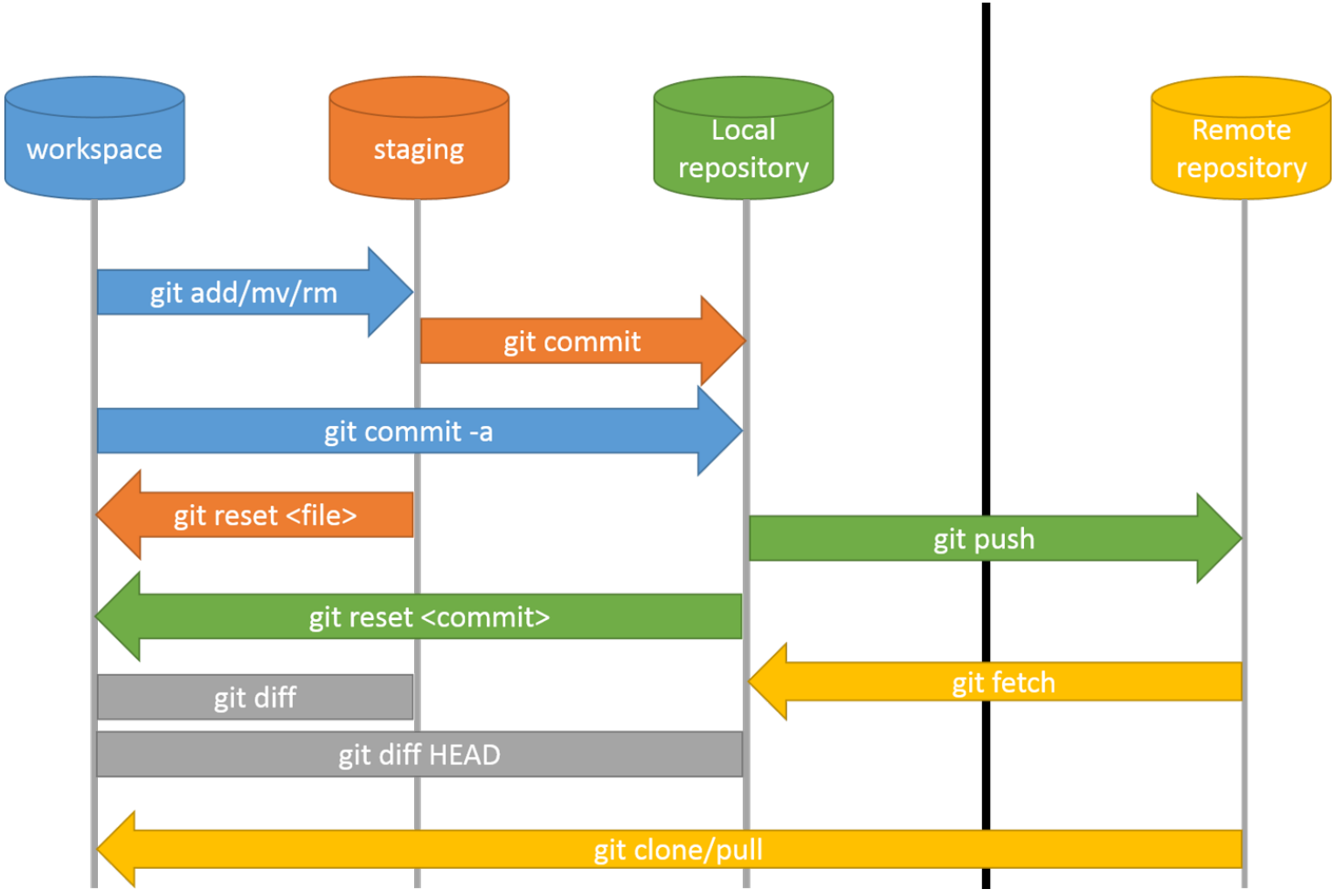


Figure: Pro Git, 2nd Edition

Version Control

Marco Morales
marco.morales@columbia.edu

GR5069
Topics in Applied Data Science
for Social Scientists

Spring 2019
Columbia University