Git / GitHub

ANU - Research School of Economics

Matthew McKay [matthew.mckay@anu.edu.au]

December 2019

What is Git?

Git is a *distributed* version control system that sits on top of your file system that tracks the state of a collection of files.

Its major strength is text type data (manuscripts, code etc.) but can keep track of other file types also.

It can help you determine exactly what has changed, why it has changed, and who has changed it.

What is GitHub?

GitHub is **not** Git.

It is a website that hosts Git projects and adds an interface for:

1. tracking issues,
2. reduce complexity of code reviews, and
3. simplifies collaboration on projects

Key Concepts of Git

1. Working Directory
2. Staging Area
3. Local Repository (Repo)
4. Remote Repository (Repo)

as **Git** is *distributed* the local and remote repositories have access to the same history information.

Git Workflow

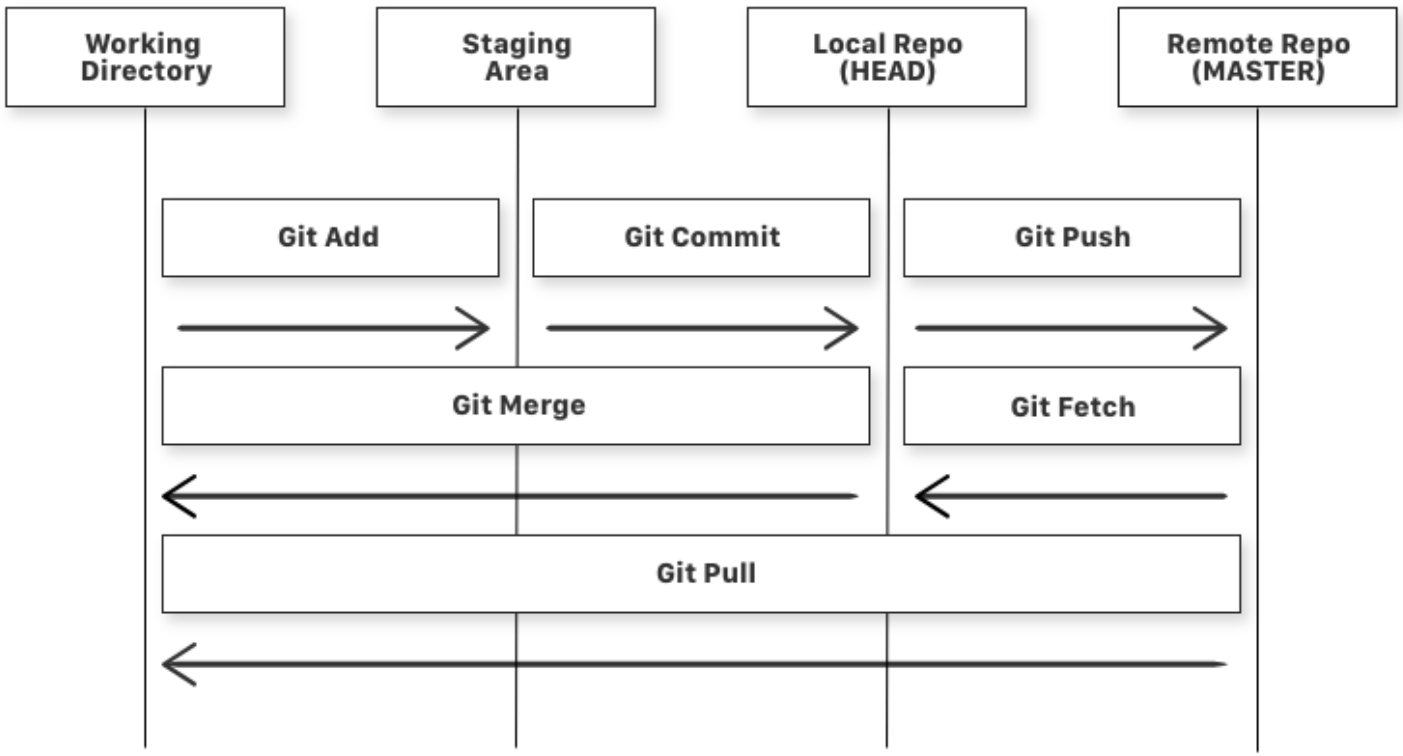


image courtesy of this tutorial

Three main states for a file

A **file** in a Working Directory can have one of the following three states

1. **Modified** changes detected
2. **Staged** changes are marked to be committed
3. **Committed** changes stored in local repository

find this out using: `git status`

Setting up Git

When using Git for the first time it can be a good idea to setup Git with the correct user information.

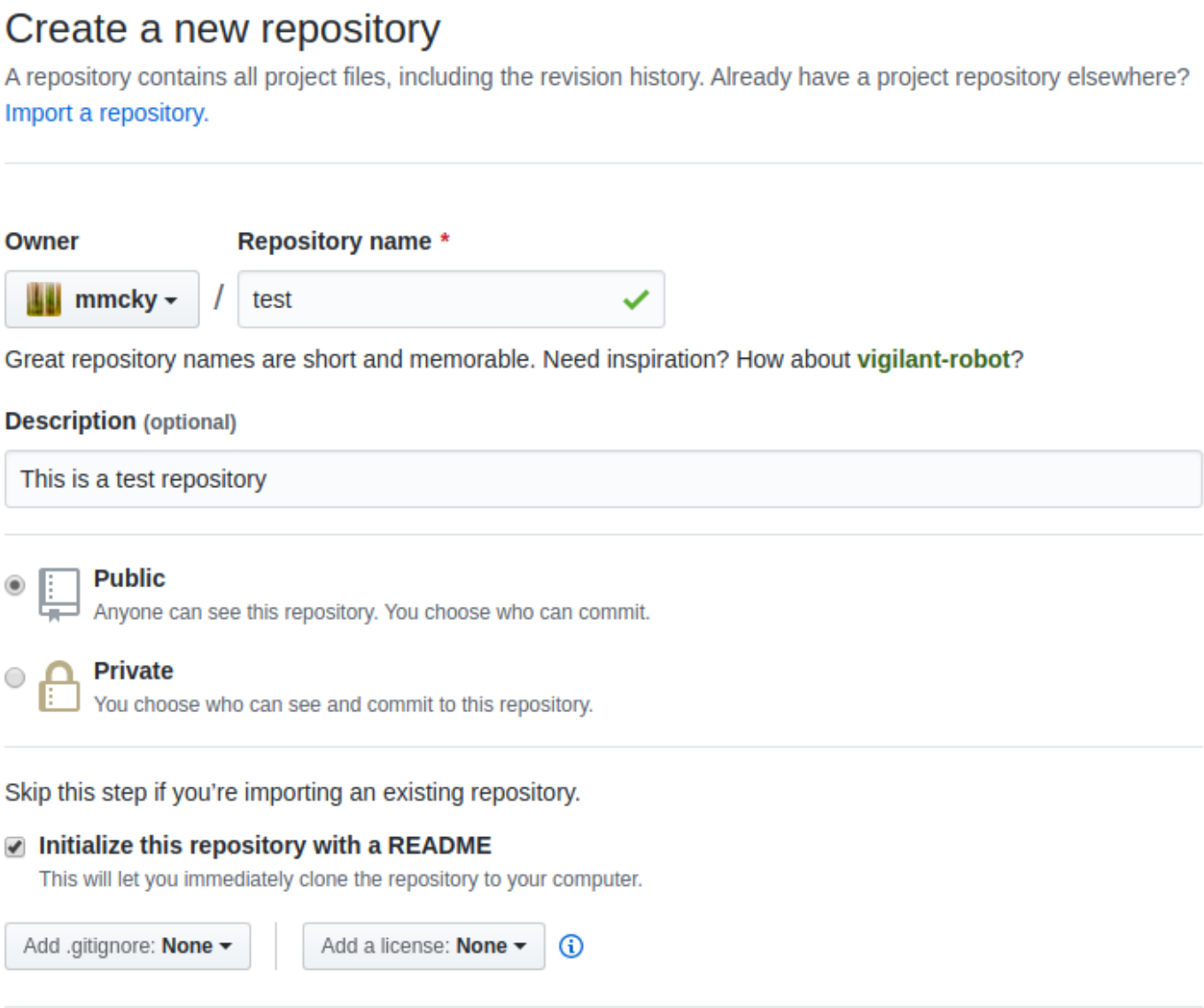
Open a terminal (or Anaconda Bash (Windows)):

```
$ git config --global user.name "Your name here"
$ git config --global user.email "your_email@example.com"
```

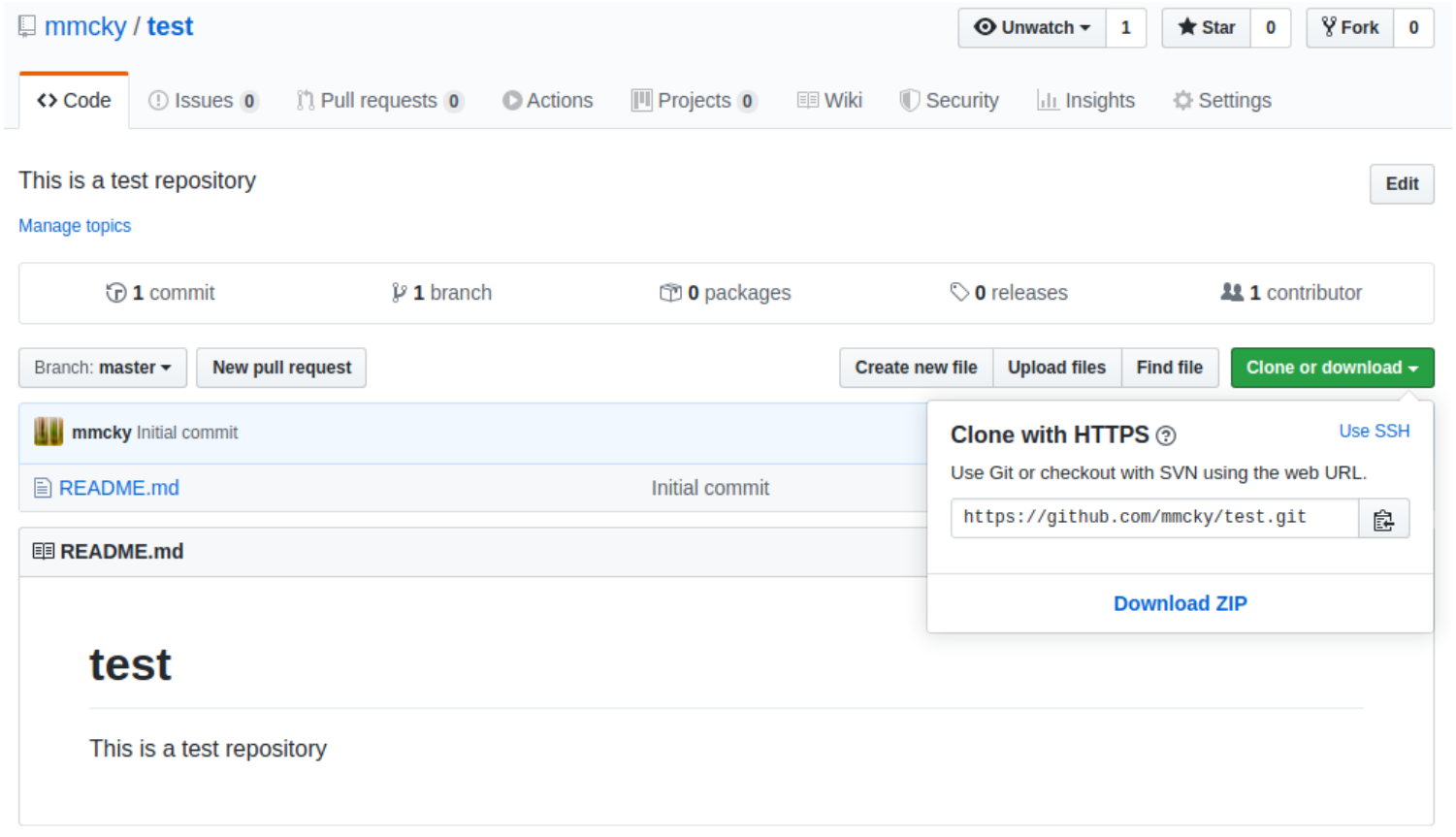
Check using: `git config --list`

Setting up a Repository (GitHub)

Go to GitHub (in browser) and click on **New**



New Repository



Click on **Clone or download**:

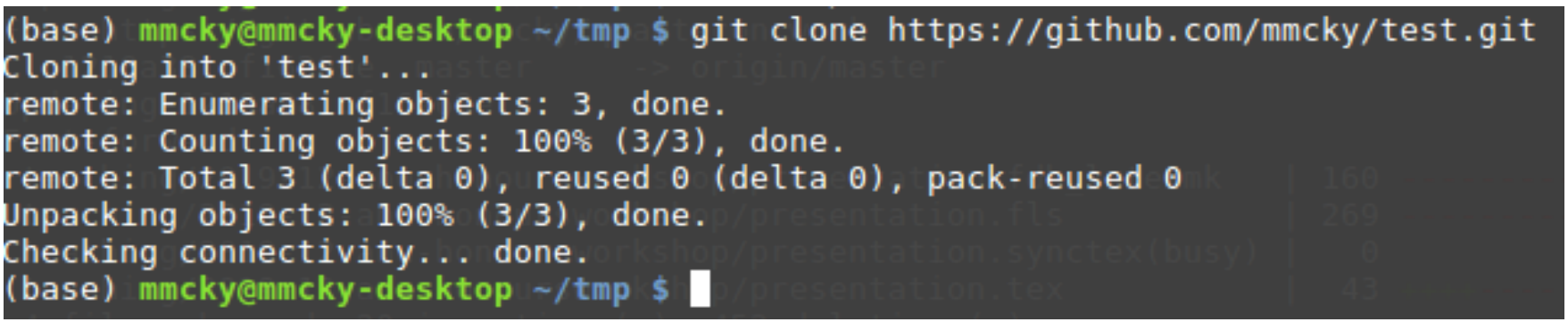
`https://github.com/<username>/test.git` and copy to clipboard

Cloning your Repository

To get a copy of the repository on your local machine

Open a **terminal**:

```
git clone https://github.com/<username>/<repo-name>.git
```



Getting the latest from GitHub

If you **already** have a local copy of the repository on your local computer then you should always fetch the latest changes before starting your work.

Open a **terminal**:

```
git pull
```

this is particularly the case in collaborative environments or if you have changed computers.

Adding a Change

Navigate to the repository `cd test`

Create a file such as **first.txt** and use `git status` to check on Git state

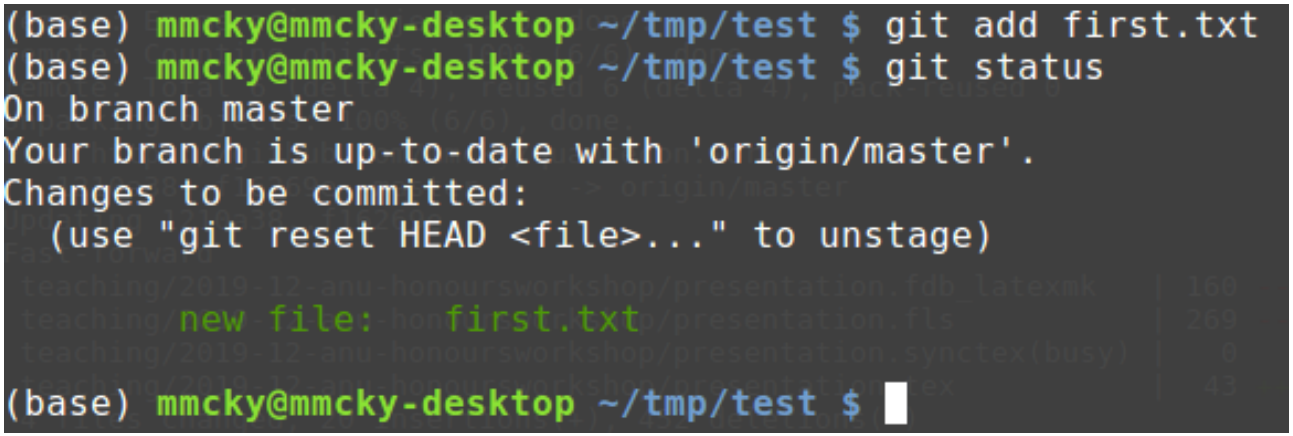
```
(base) mmcky@mmcky-desktop ~/tmp/test $ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        first.txt

nothing added to commit but untracked files present (use "git add" to track)
(base) mmcky@mmcky-desktop ~/tmp/test $
```

Commiting a Change (Local) - Part 1

We will want to move the file **first.txt** into the staging area in preparation for committing.

In Terminal:

```
git add first.txt
```



Commiting a Change (Local) - Part 2

Once we have added the files we want to commit we can commit them

In Terminal:

```
git commit -m "initial commit"
```

```
(base) mmcky@mmcky-desktop ~/tmp/test $ git commit -m "initial commit"
[master 23ec3fc] initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 first.txt
(base) mmcky@mmcky-desktop ~/tmp/test $ git log
commit 23ec3fcc035c64b898639cba12b23c275c1e290e
Author: Matt McKay <mamckay@gmail.com>
Date:   Thu Dec 12 09:28:59 2019 +1100

    initial commit

commit 35a814e4797d3b7e33369d35e85800349164c1bf
Author: mmcky <mmcky@users.noreply.github.com>
Date:   Thu Dec 12 09:09:07 2019 +1100

    Initial commit
(base) mmcky@mmcky-desktop ~/tmp/test $
```

Pushing a Change (Remote)

To get those changes on your remote GitHub copy of the Repository we need to push them to GitHub:

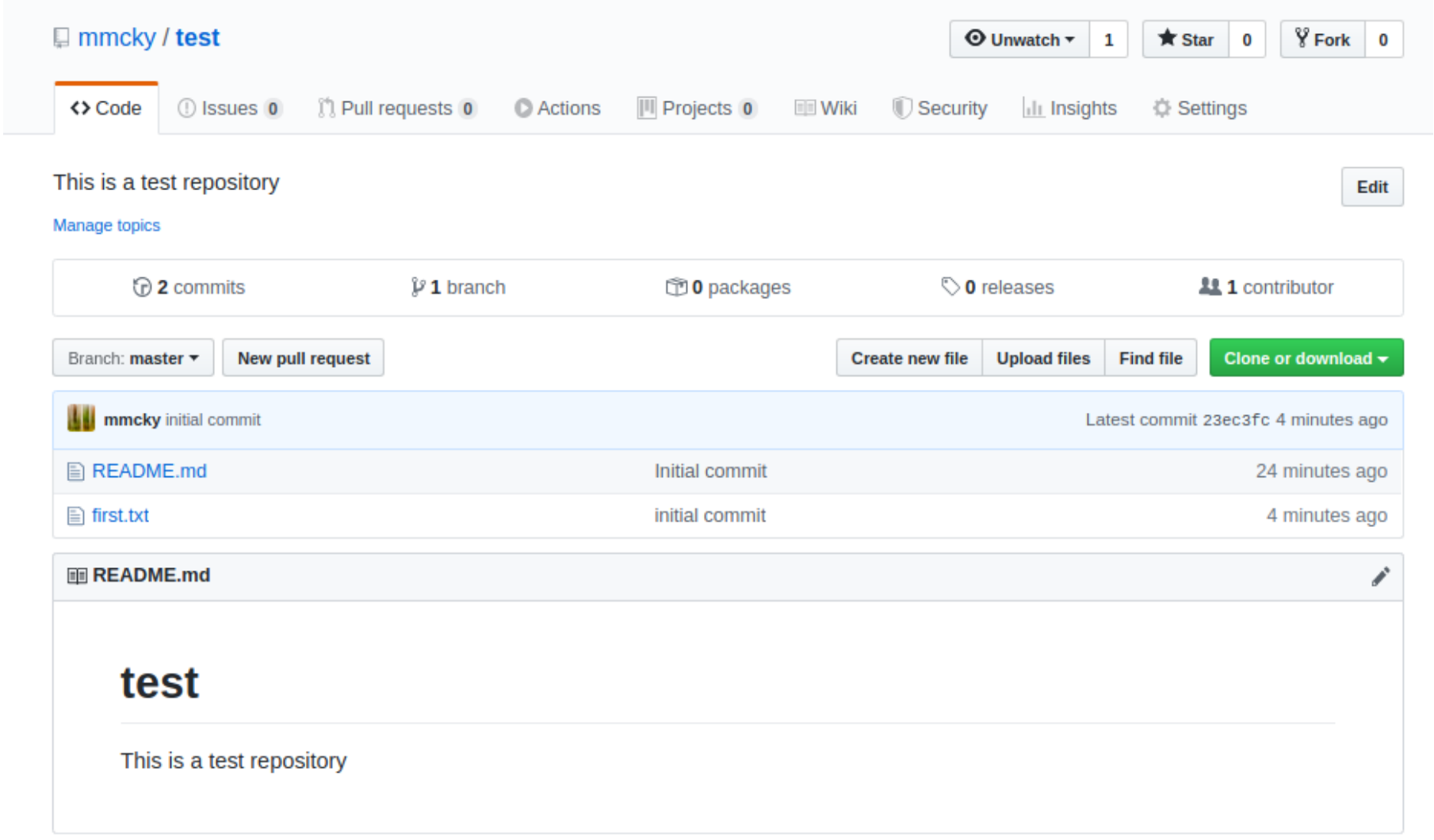
In Terminal:

```
git push origin master
```

```
(base) mmcky@mmcky-desktop ~/tmp/test $ git push origin master
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 297 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/mmcky/test.git
   35a814e..23ec3fc  master -> master
(base) mmcky@mmcky-desktop ~/tmp/test $
```

Note: on collaborative projects you rarely push to **master**

Viewing results on GitHub



More Advanced Git Features

Git has a **lot** of features:

1. Forking
2. Branching
3. Rebasing
4. Tagging
5. Stashing
6. ...

we will cover **branching**

Branching

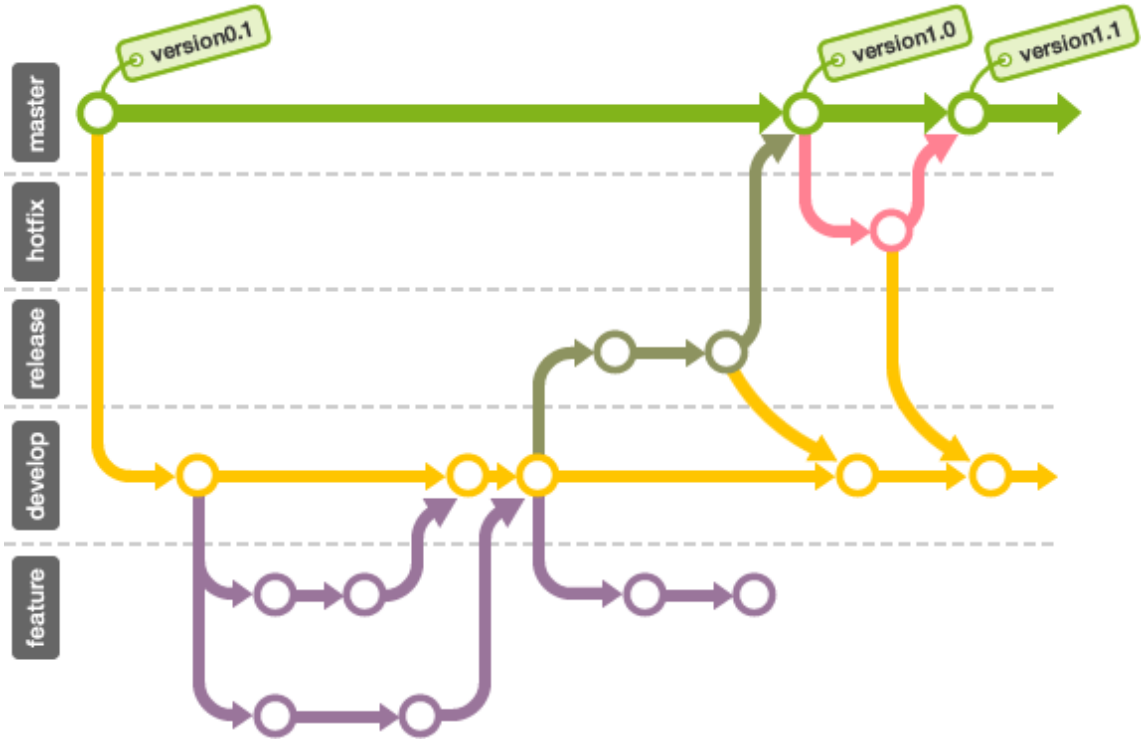
A **branch** is like a parallel copy of the repository.

It allows development to occur in parallel of the **master** branch.

The master branch does not get modified until the branch is merged into the **master** branch.

Using GitHub this is called a **pull request**

Branching


version0.1
version1.0
version1.1
master
hotfix
release
develop
feature

Setup a Branch

Let's setup a branch

```
git checkout -b first-branch
```

```
(base) mmcky@mmcky-desktop ~/tmp/test $ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working directory clean
(base) mmcky@mmcky-desktop ~/tmp/test $ git checkout -b first-branch
Switched to a new branch 'first-branch'
(base) mmcky@mmcky-desktop ~/tmp/test $ git status
On branch first-branch
nothing to commit, working directory clean
(base) mmcky@mmcky-desktop ~/tmp/test $
```

and add a file **second.txt**

Adding a Change

We now want to follow the same **commit** workflow as before

```
git add second.txt
git commit -m "adding a second file"
```

```
(base) mmcky@mmcky-desktop ~/tmp/test $ git status
On branch first-branch
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        second.txt

nothing added to commit but untracked files present (use "git add" to track)
(base) mmcky@mmcky-desktop ~/tmp/test $ git add second.txt
(base) mmcky@mmcky-desktop ~/tmp/test $ git status
On branch first-branch
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   second.txt

(base) mmcky@mmcky-desktop ~/tmp/test $ git commit -m "adding second file to repo"
[first-branch e9478a1] adding second file to repo
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 second.txt
(base) mmcky@mmcky-desktop ~/tmp/test $ git status
On branch first-branch
nothing to commit, working directory clean
(base) mmcky@mmcky-desktop ~/tmp/test $
```

Pushing the Change

Pushing this branch to GitHub's copy of the **test** repo

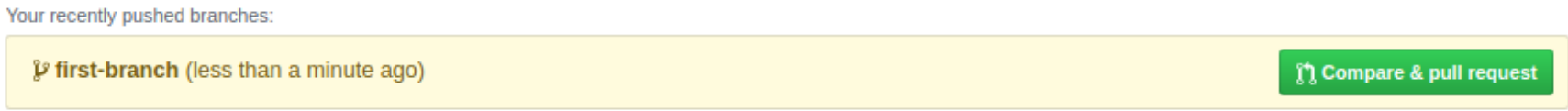
```
git push origin first-branch
```

```
(base) mmcky@mmcky-desktop ~/tmp/test $ git status
On branch first-branch
nothing to commit, working directory clean
(base) mmcky@mmcky-desktop ~/tmp/test $ git push origin first-branch
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 318 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'first-branch' on GitHub by visiting:
remote:      https://github.com/mmcky/test/pull/new/first-branch
remote:
To https://github.com/mmcky/test.git
 * [new branch]      first-branch -> first-branch
(base) mmcky@mmcky-desktop ~/tmp/test $
```

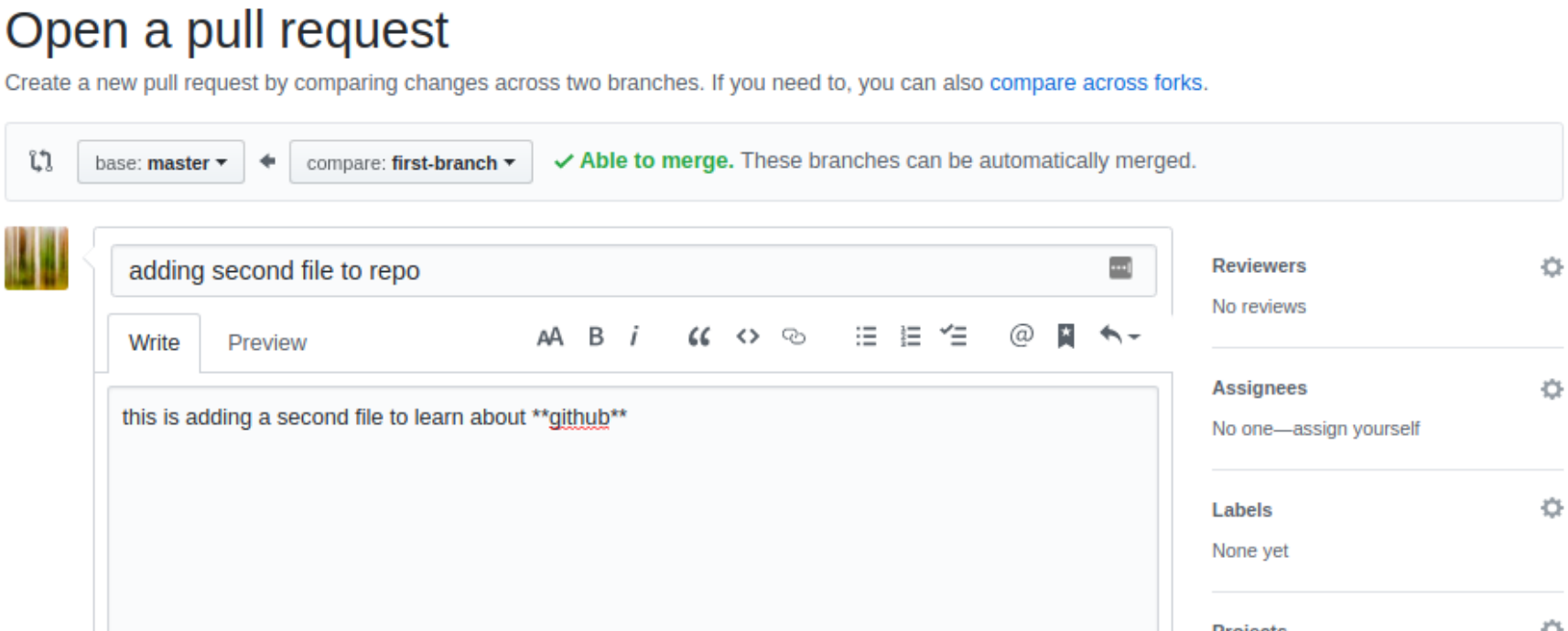
Note: *origin* is set to GitHub and we want to push the branch to it

Setup a PR on GitHub

Look for:



then setup a **PR** by clicking on **Compare and pull request**



Key Git Commands

98% of typical interactions with git is focused on these commands

1. **git status**
2. **git pull**
3. **git add**
4. **git commit -m ”message”**
5. **git push**
6. **git checkout -b <branch-name>**

The Standard Terminal Workflow

1. `git pull` : retrieves new updates
2. `git status` : show status of commit
3. General work and edits
4. `git status` : show status of commit
5. `git add .` : puts **all** changed files into **staging area**
6. `git commit -m "some message"` : commit changes to local git repository with a description
7. `git push` : pushes the new version to Github

the workflow is altered slightly when using **branches** by

1. creating a branch to do the work in
2. pushing that branch to GitHub for review and merging

Other Git Commands and Concepts

Git has a **lot** of commands not covered here

1. **git clone**
2. **git branch**
3. **git merge**
4. **git rebase**
5. **git log**
6. **forking** used in open source work
7. **git checkout hash**
8. **git diff hash1 hash2**

Git Resources

Understanding Git:

1. Github Tutorials
2. Git Book
3. Blog: Beginners
4. Blog: Git Guide
5. Blog: Git Quickstart
6. Blog: Useful Git Commands

GUI Options (OS X and Windows):

Some like to start by using a GUI

1. Github Desktop
2. Git Fork
3. Git Kraken

I would recommend *terminal based workflows*

Exercise 1: Submit first Homework PR

Setup a branch and pull request for course repository for homework submission

```
git clone https://github.com/QuantEcon/summer_course_2019.git
```

1. Get updated version of the Repository: `git pull`
2. Setup a New Branch: `git checkout -b add-<username>`
3. Add your own folder **"<github username>"** to the **homework** folder
4. Add a README file: `git add README.md`
5. Look at the git status: `git status`
6. Make a commit: `git commit -m "<message>"`
7. Push branch to GitHub: `git push origin add-<firstname>`
8. Setup a Pull Request

Your branch name should be something like `add-mmcky` to avoid collisions.