Version Control with Git

Kaylea Nelson

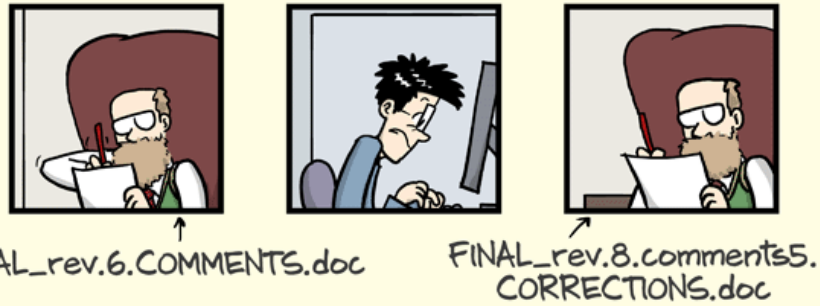
research.computing@yale.edu

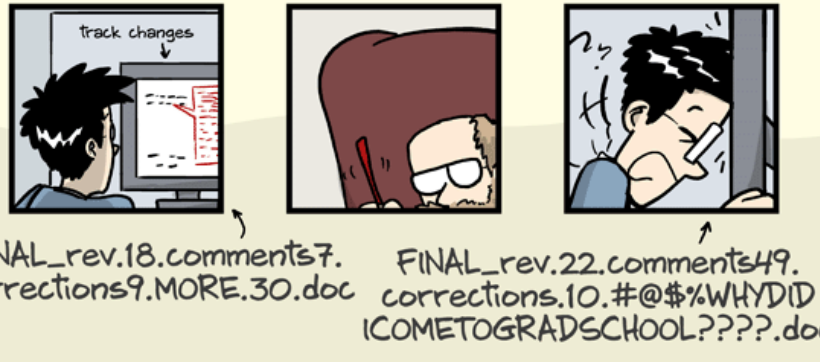
Yale *Center for Research Computing*

Version Control with Git

- What is Version Control and Git?
- Putting Your Code into Git
- Connecting Your Repository to Bitbucket
- Utilizing Your Repository's History
- Collaboration: Merging and Conflicts

"FINAL".doc





What is Version Control?

"The whole idea behind any version control system is to store "safe" copies of a project so that you never have to worry about irreparably breaking your code base."

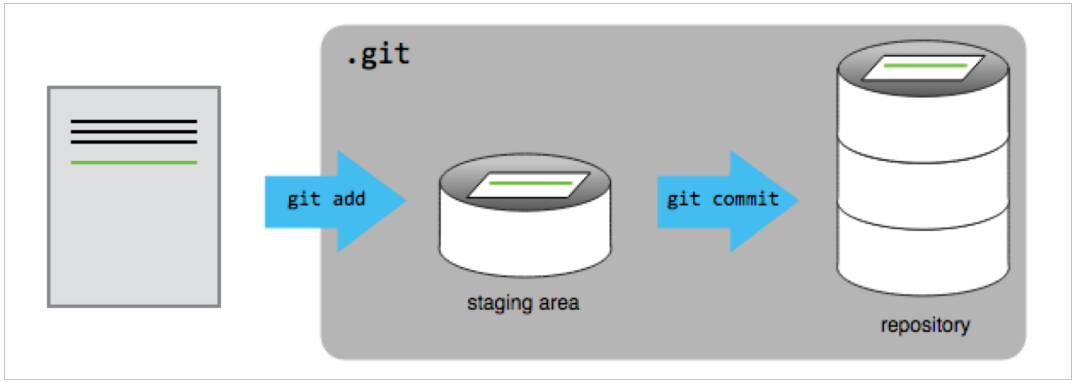
– Bitbucket.org

- Easy and powerful way to track changes to your work
- Useful for both writing (if using e.g. LaTeX) and code
- Backups of your work
- General coding safety net

What is Git? How does it work?

Git tracks changes to a file (or set of files) through a series of snapshots called “commits” or “revisions”.

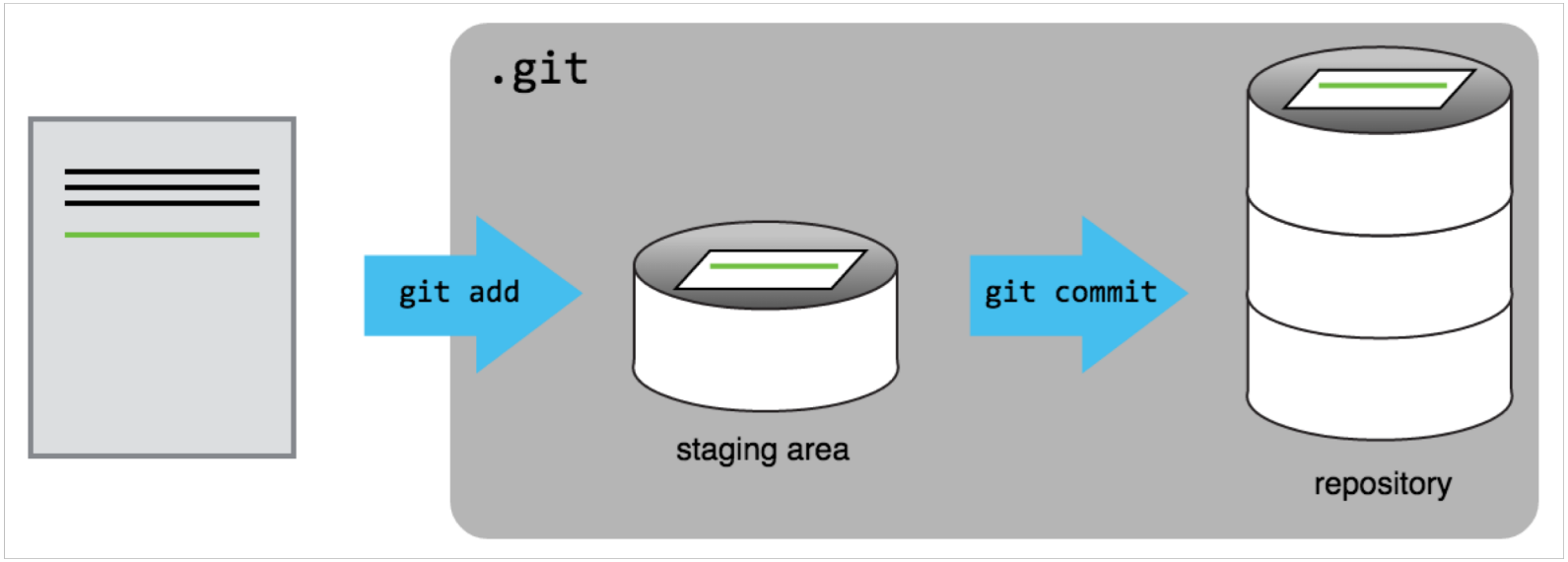
These snapshots are stored in a “repository” which contains a history of all the changes to the files.



How is Git useful to me?

- “Why isn’t it working all of a sudden?”
- Cleaner file system (no more “code, codev2, codev3_test, codev3_test1” directories)
- Record of your edits (and thought process!)
- Check for bugs in inconsistent results
- Unlimited and powerful “undo”
- Collaboration!

Putting Your Code into Git



Configure Git

- Global configurations for Git

```
$ git config --global user.name "Your Name"
$ git config --global user.email "your.email@yale.edu"
```

Setup Repository

- Initialize repository

```
$ git init
```

This create a `.git` directory in your directory that contains all the version control information. DO NOT DELETE!!!

```
$ ls -a
.               ..              .git
```

Add Existing Files to Repository

- The Git repository can be initialized before or after you create any files. To version control existing files, just add them to the repository.

```
$ git add myplot.py
```

Check Status of Repository

```
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   myplot.py

Untracked files:
  (use "git add <file>..." to include in what will
be committed)

    myfigure.png
```

Make Initial Commit

- Now we want to permanently save the changes to repository, an action called “committing”

```
$ git commit –m "initial commit"
```

Leaving Files Out of Repository

- You don’t want to add every file to your repository. The good rule of thumb is to exclude files if they are a product of your code. Examples of files to exclude:
  - Image files
  - PDFs
  - Compiled code (including .o or .pyc files)
  - System files (e.g., .DS_Store)

Automate Exclusions

- To easily automate exclusions, create a `.gitignore` file.

```
$ cat .gitignaore
.DS_Store
*.png
*.pyc

$ git add .gitignore
$ git commit –m "added .gitignore"
```

- Now these files won't show up as "untracked" in the `git status` command and can't accidentally get added to the repository

Make Changes!

- Make changes to the file and then check on the repository

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be
committed)
  (use "git checkout -- <file>..." to discard
changes in working directory)

    modified:   myplot.py

no changes added to commit (use "git add" and/or
"git commit -a")
```

Make Changes!

- Add and commit your changes

```
$ git add myplot.py
$ git commit -m "increased frequency"
[master 21e2dd2] increased frequency
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Review Changes

- You can check to see what has been modified before adding files using `git diff`

```
$ git diff
diff --git a/myplot.py b/myplot.py
index 3c179cc..3eb9a45 100644
--- a/myplot.py
+++ b/myplot.py
@@ -2,7 +2,7 @@ import matplotlib.pyplot as plt
 import numpy as np

 t = np.arange(0.0, 2.0, 0.01)
-s = np.sin(2*np.pi*t)
+s = np.sin(4*np.pi*t)

 plt.plot(t, s)
 plt.xlabel('time (s)')
```

Review Changes

- You can check to see what has been modified before committing using `git diff --staged`

```
$ git diff --staged
diff --git a/myplot.py b/myplot.py
index 3c179cc..3eb9a45 100644
--- a/myplot.py
+++ b/myplot.py
@@ -2,7 +2,7 @@ import matplotlib.pyplot as plt
 import numpy as np

 t = np.arange(0.0, 2.0, 0.01)
-s = np.sin(2*np.pi*t)
+s = np.sin(4*np.pi*t)

 plt.plot(t, s)
 plt.xlabel('time (s)')
```

Writing a Good Commit Message

- The commit message should be a high level explanation of the change
  - Don’t be too brief
  - Also, don’t exactly quote the change
- Example:
  - Bad: “Changes”
  - Bad: “Changed line 178 in plot_bM_vs_t.py”
  - Better: “Change color of pressure line to red”
- Most important question: If you are looking at this message in 6 months, is it going to make sense and be useful?

Other Useful Commands

- Rename or move a file in the repository

```
$ git mv <old_filename> <new_filename>
```

- Delete a file from the repository

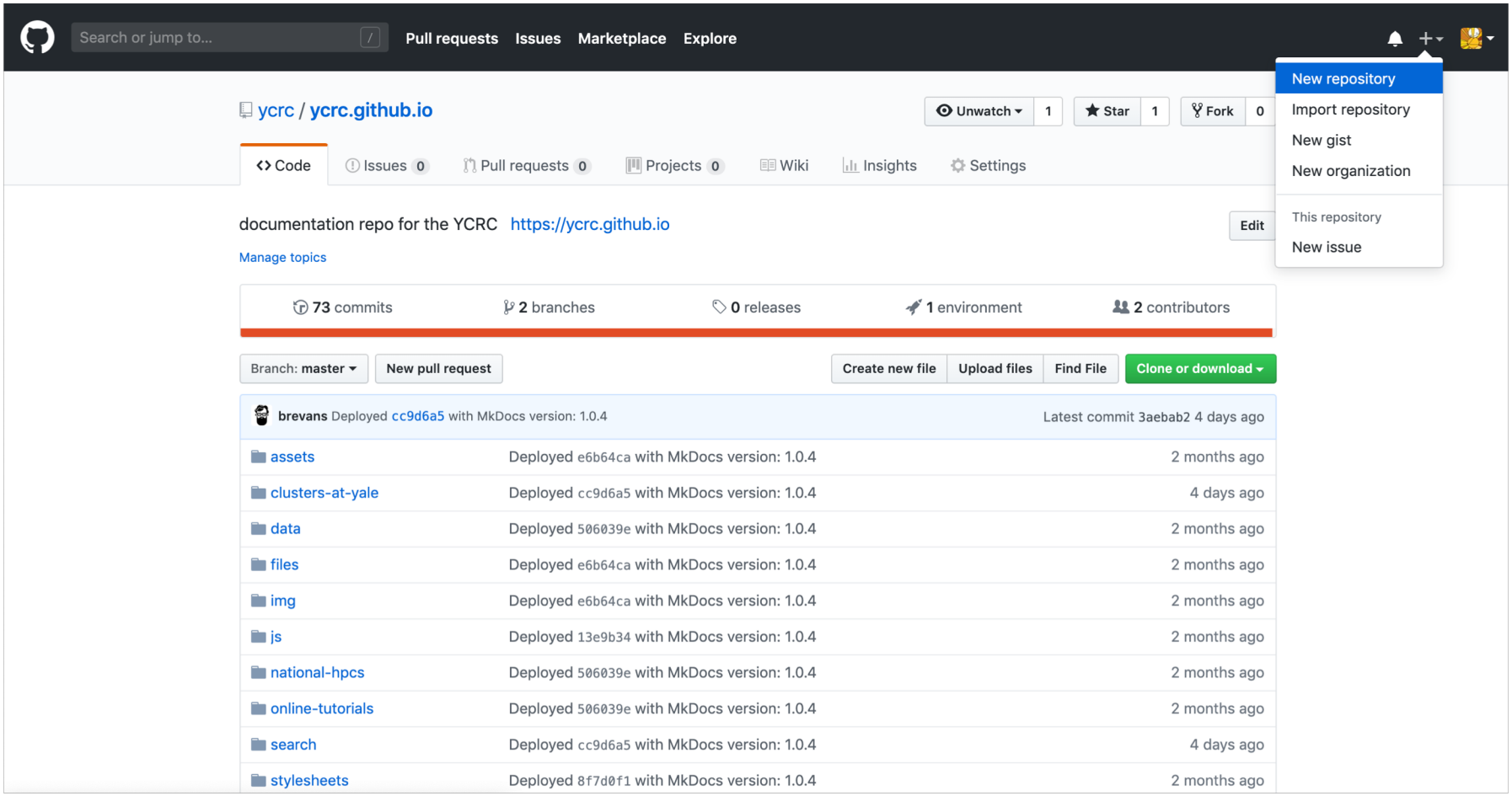
```
$ git rm <filename>
```

Connecting Your Repository to Github

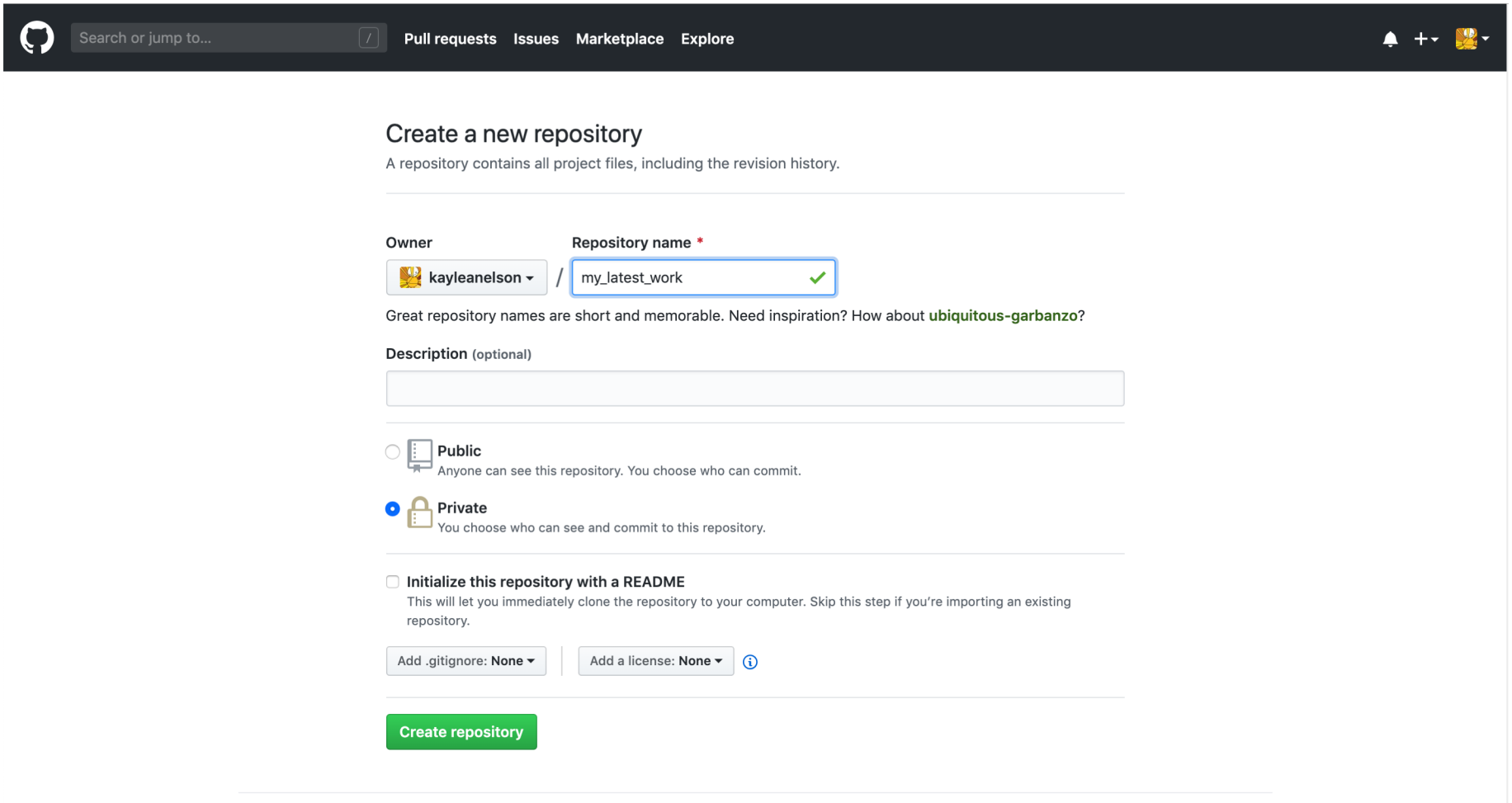
1. Create repository on your online account
2. Follow included instructions to get your local repository connected to your remote repository
3. Push committed changes to the remote repository

```
…
$ git commit -m "<message>"
$ git push
```

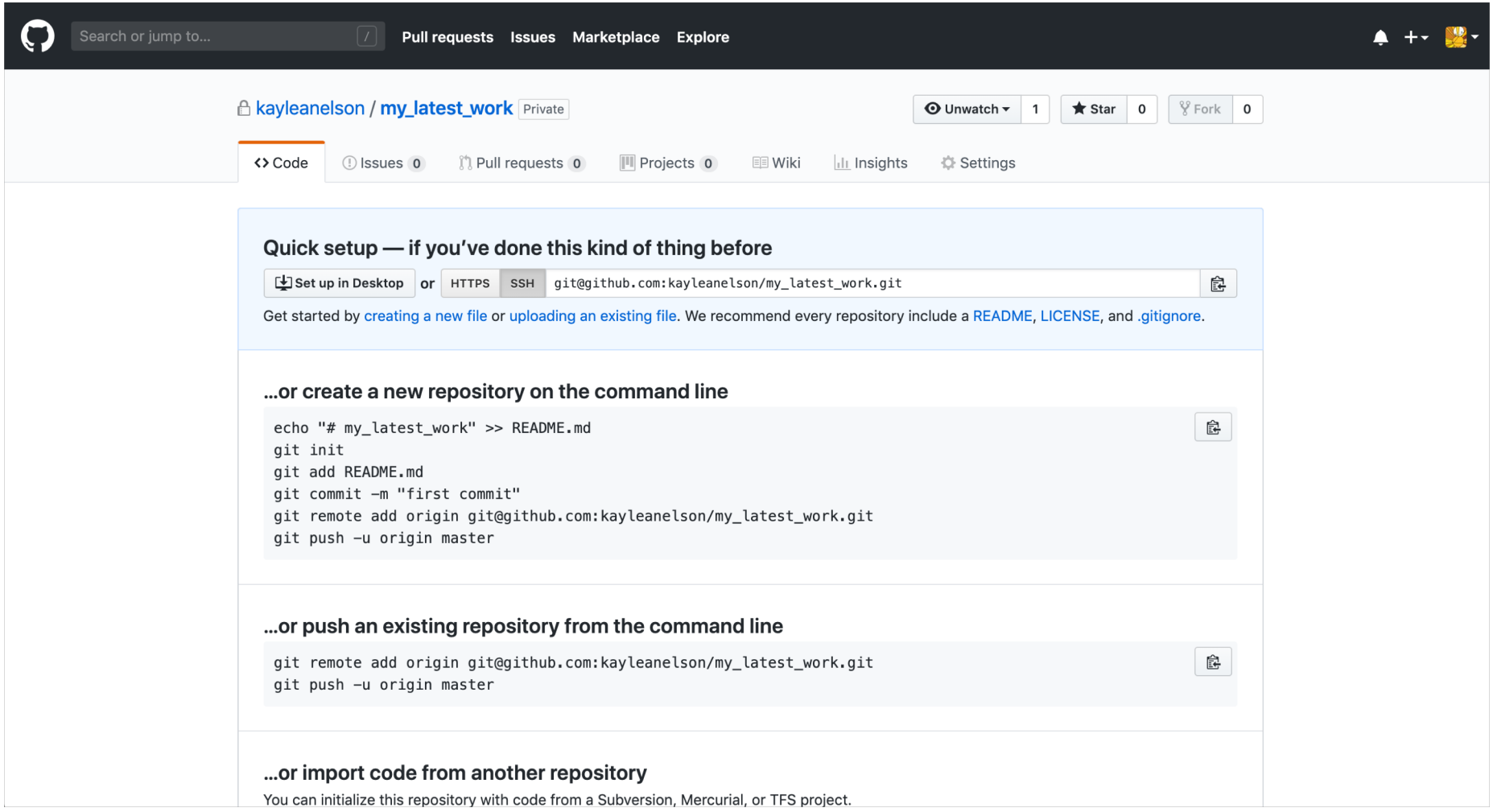
Create a New Repository on Github



Create a New Repository on Github



Follow the Instruction to Push



Push Future Commits

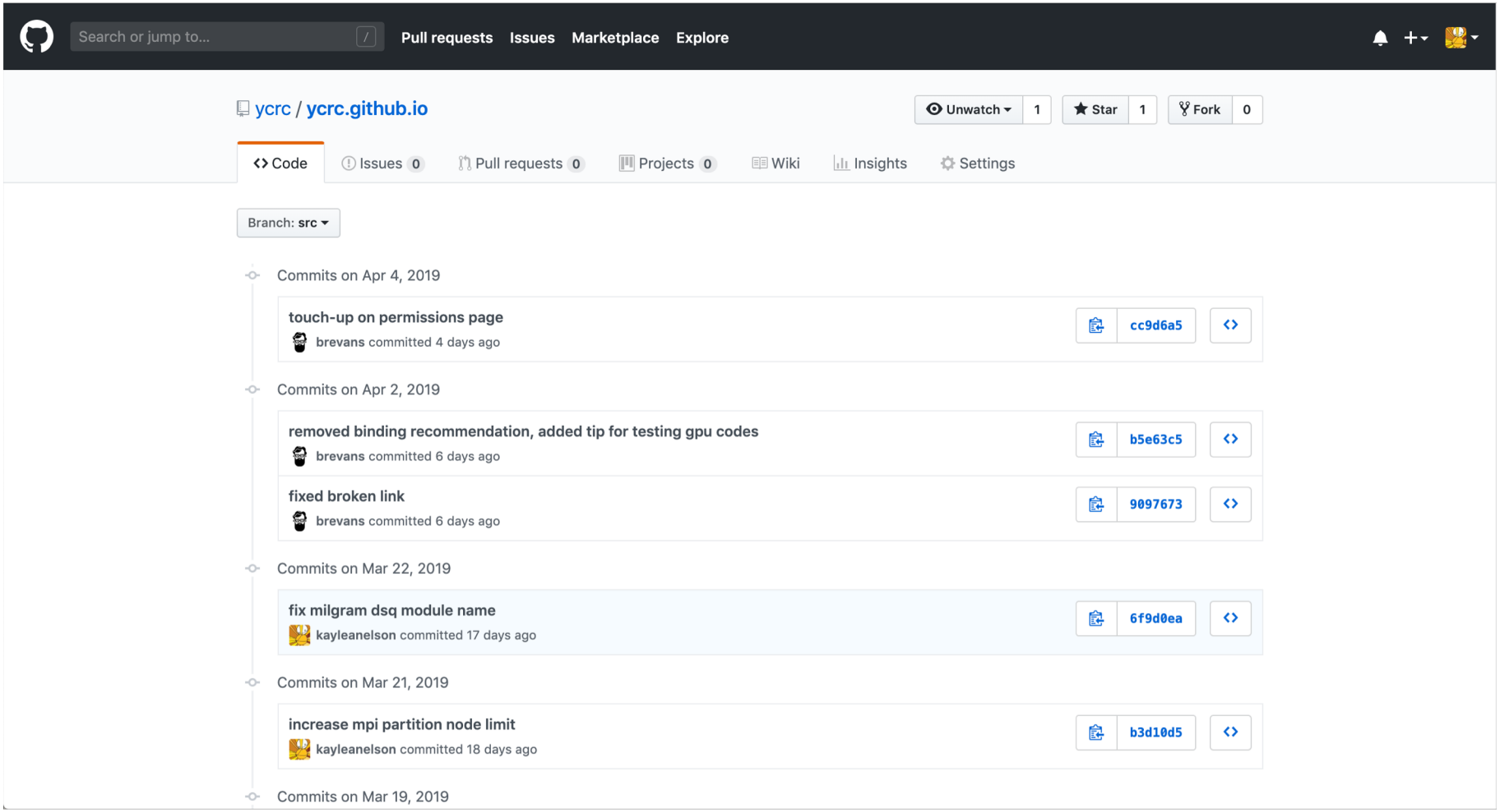
- After the initial “push” to the remote repository, just remember to push any new commits and you will have easy remote backups of your work!

```
…
$ git commit -m "<message>"
$ git push
```

Remote Repository Hosts Options

- github.com – unlimited free public and private repos for everyone
- Bitbucket.org – unlimited repos [but be careful signing up with yale.edu email]
- git.yale.edu – free fully featured accounts. Only available on Yale network or VPN and with Yale netid

Utilizing Your Repository's History



Review History

- You can see a history of all recent commits
  - Detailed Log:

```
$ git log
$ git log -1
```

  - Simplified Log

```
$ git log --oneline
$ git log --oneline --graph --decorate
```

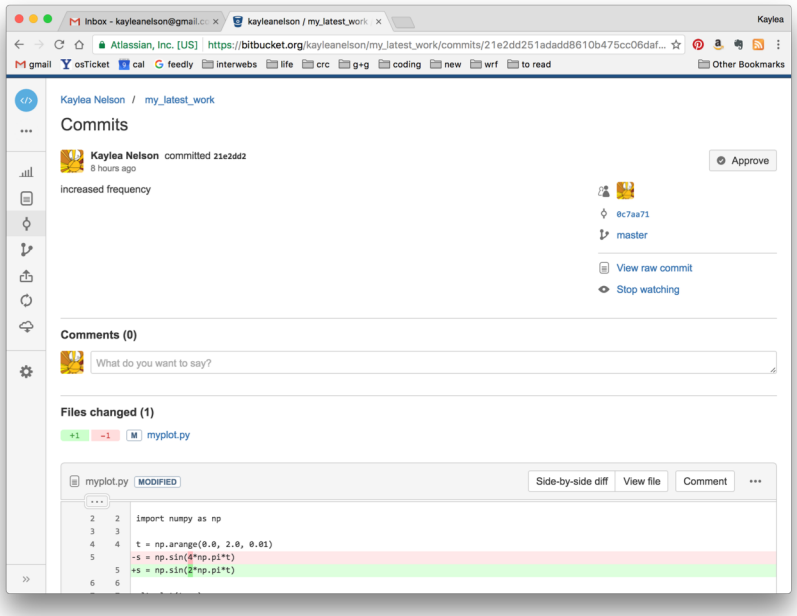
Compare Revisions

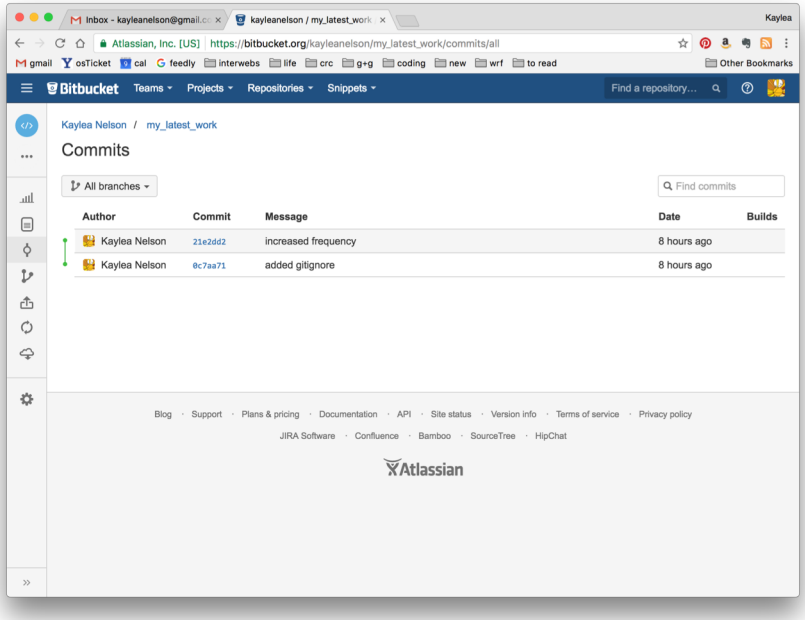
- You can compare two revisions to see what changes were made with `git diff`

```
$ git diff HEAD 0c7aa71
diff --git a/myplot.py b/myplot.py
index 3c179cc..3eb9a45 100644
--- a/myplot.py
+++ b/myplot.py
@@ -2,7 +2,7 @@ import matplotlib.pyplot as plt
 import numpy as np

 t = np.arange(0.0, 2.0, 0.01)
-s = np.sin(2*np.pi*t)
+s = np.sin(4*np.pi*t)

 plt.plot(t, s)
 plt.xlabel('time (s)')
```

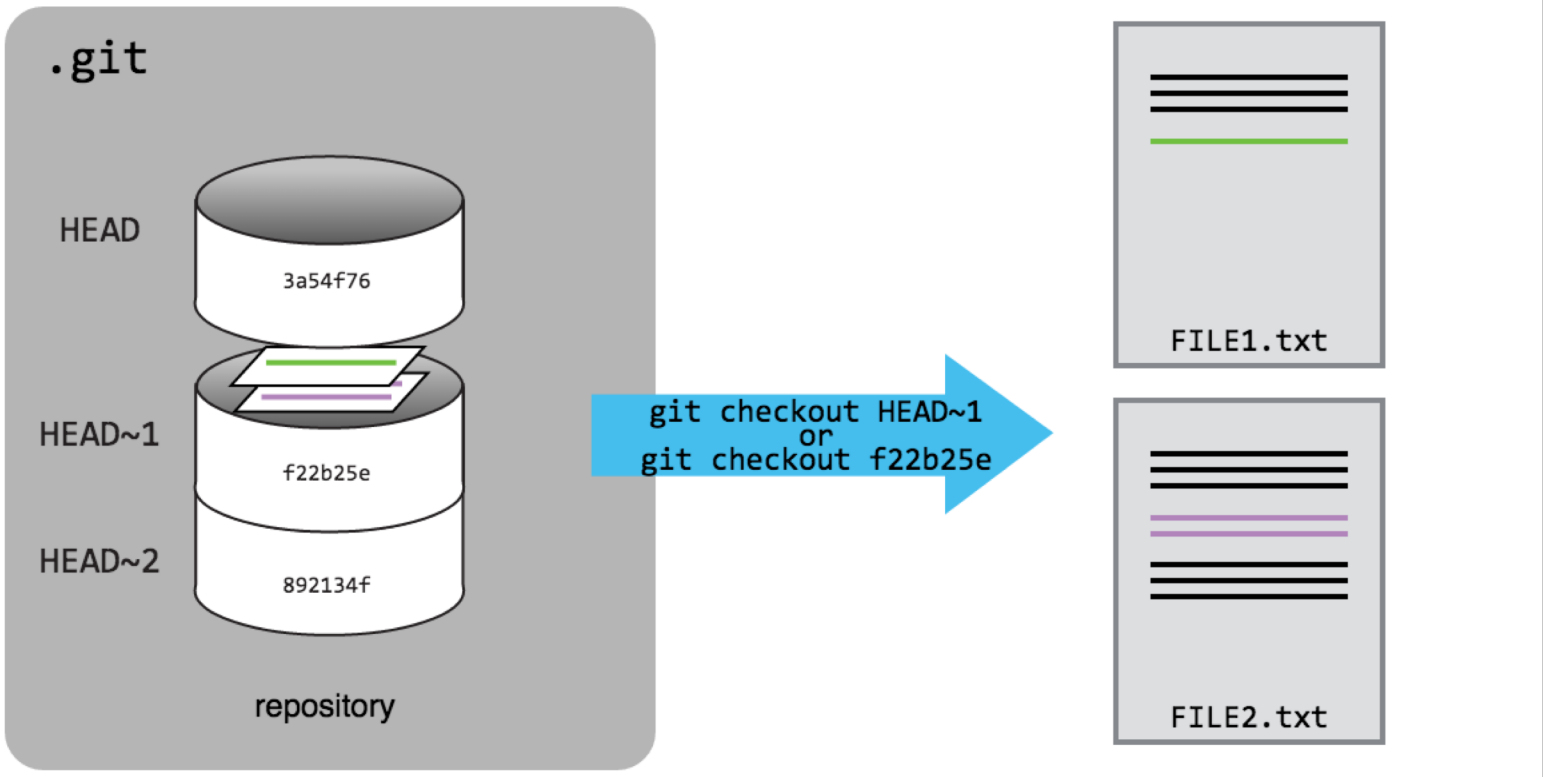
Review History

- The interfaces on Github and Bitbucket are also great for exploring the commit history and tracking changes





Checkout Previous Commits



Checkout Previous Commits

- After, you have identified the revision you need to revert to, “checkout” that revision

```
$ git checkout <revision>
```

- Or just a specific file from that revision

```
$ git checkout <revision> <filename>
```

Warning: If you checkout from an old revision, any uncommitted changes to the project will be lost.

Throwaway All New Changes

- Revert your working directory to the last commit

```
$ git reset --hard
```

Warning: Any uncommitted changes to the project will be lost.

Getting Your Code in a New Location

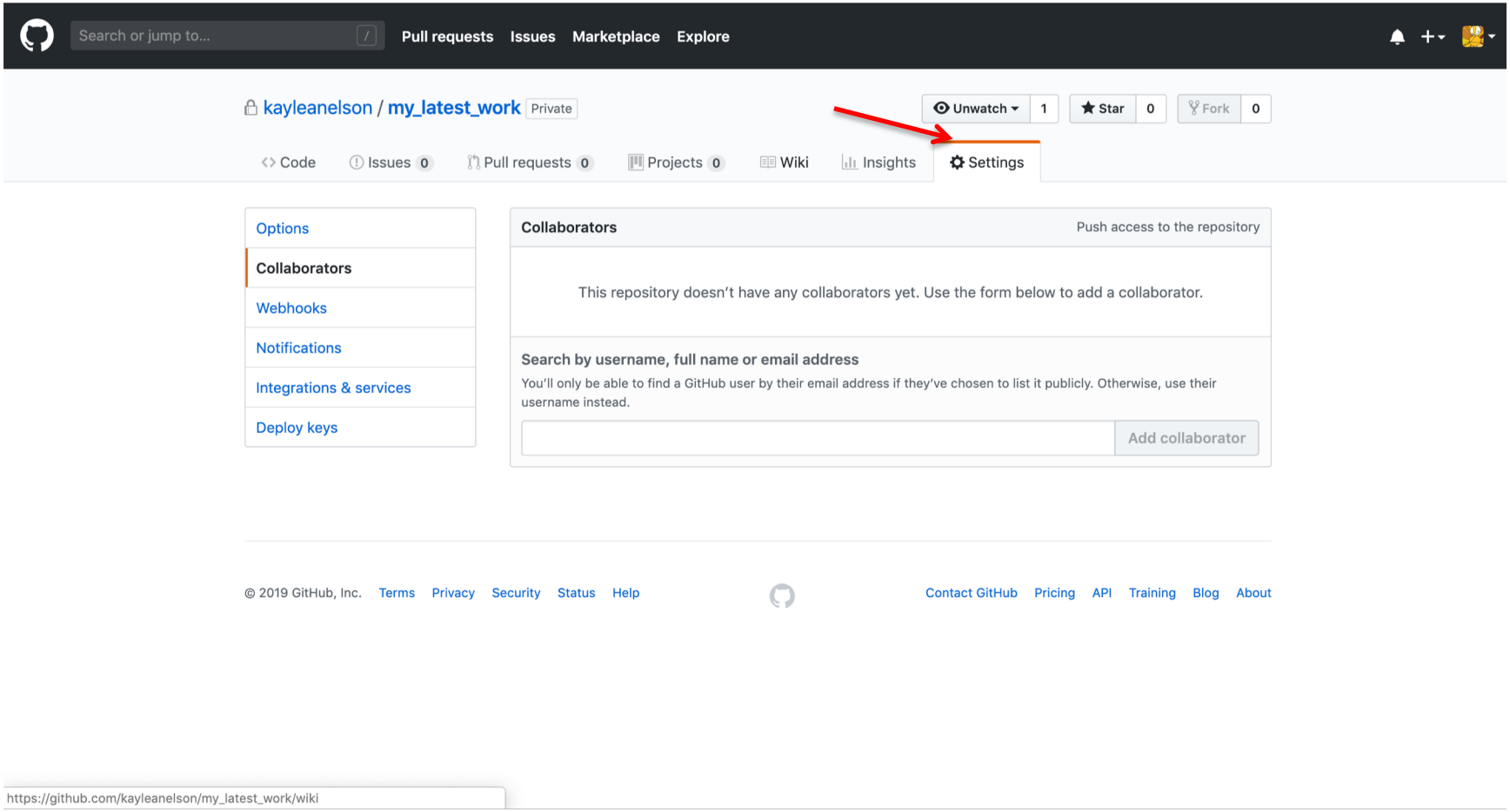
- If you have a remote repository, you can "clone" it to a new location to continue your work (e.g., copying code to the cluster, recovering your code to a new laptop)

```
$ git clone
https://kayleanelson@github.com/kayleanelson/my_latest
_work.git
```

Collaboration

- Once your work is in a remote repository, it is very easy to being to collaborate with others
  - Git has a sophisticated system for managing multiple people editing the same code base through “merging”
- Usage Examples
  - Multiple collaborators on a code
  - LaTeX papers!

Sharing Your Repository

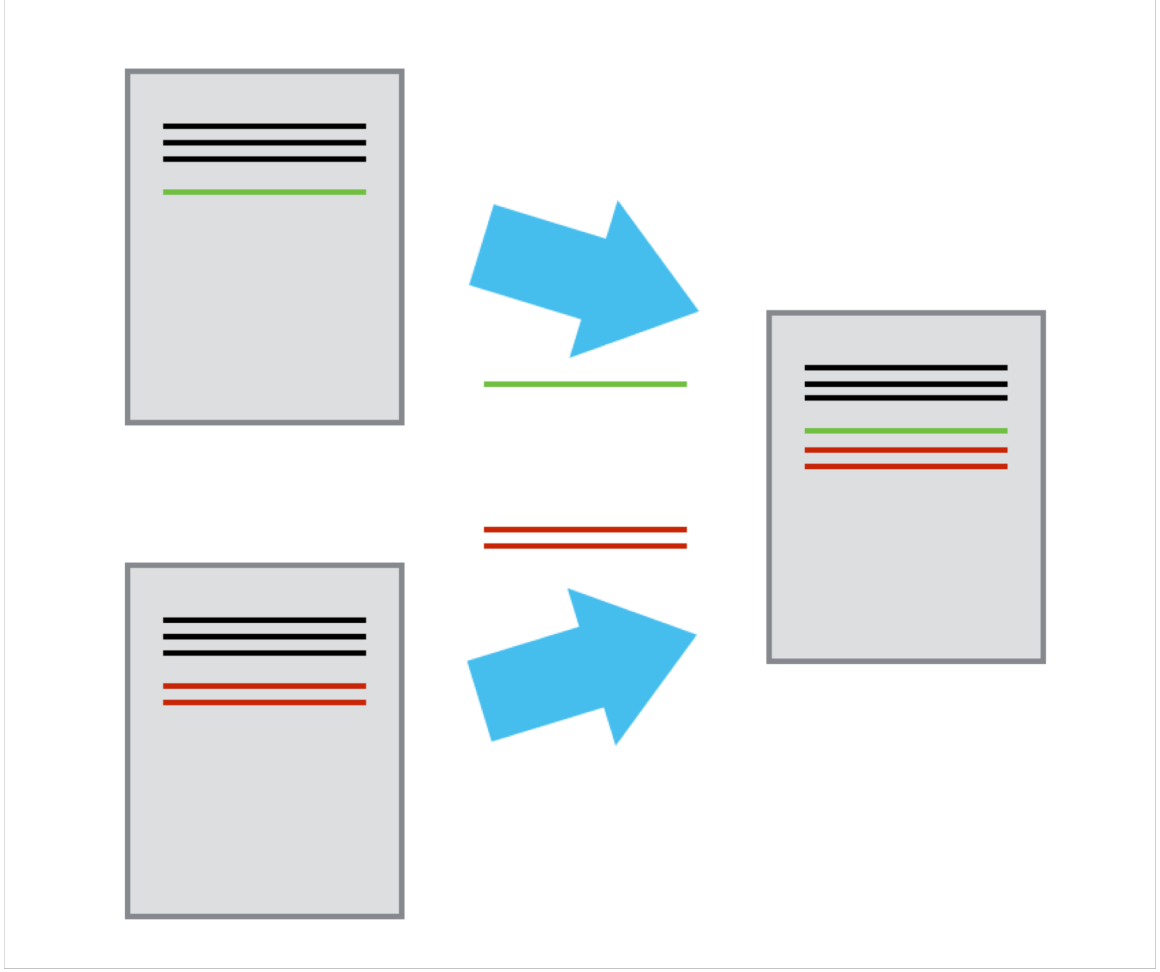


Basic Collaborative Workflow

- Pull down new commits
- Make your edits
- Add your modified files and commit
- Push commits to remote

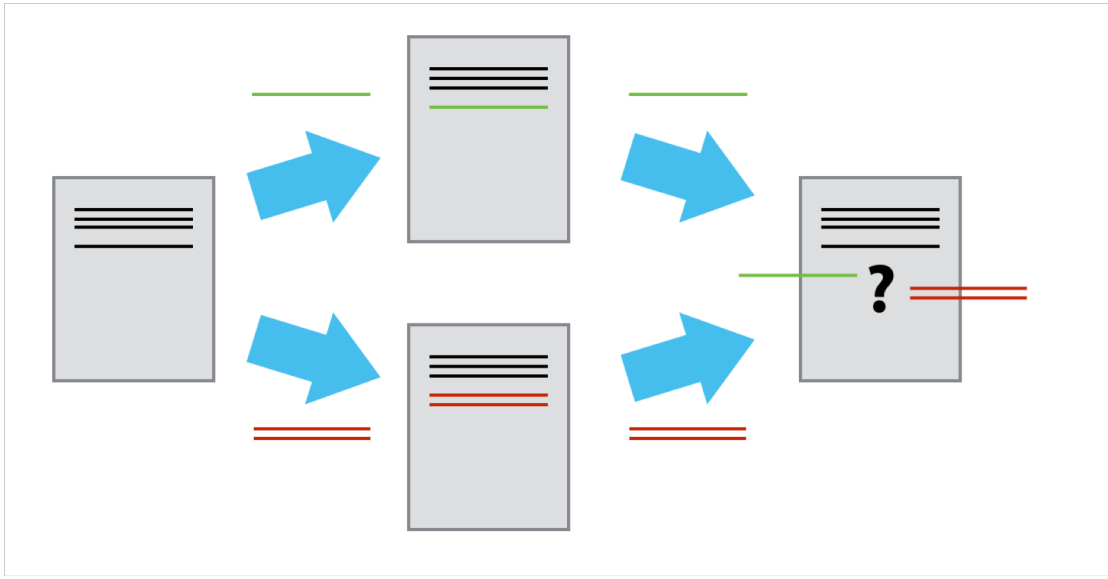
```
$ git pull
…
$ git add <files>
$ git commit –m <message>
$ git push
```

Merging



Conflicts

- Inevitably, you and your collaborator will commit overlapping changes to a file. This will create a “merge conflict”.



Resolving Conflicts

- Pull in commits and Oops!

```
$ git pull
Auto-merging myplot.py
CONFLICT (content): Merge conflict in myplot.py
Automatic merge failed; fix conflicts and then
commit the result.
```

Resolving Conflicts

- Git marks the conflicted line in the file

```
$ cat myplot.py
import matplotlib.pyplot as plt
import numpy as np

t = np.arange(0.0, 2.0, 0.01)
<<<<<<< HEAD
s = np.sin(3*np.pi*t)
=======
s = np.sin(4*np.pi*t)
>>>>>>> 7232b521f34cf3deed50f4d8aac6260616683ddf
```

- Manually merge the code in a text editor and commit the changes

Uncommitted Conflicts

- Git will also complain if you pull in changes to a file you have modified but not committed. You have two options.
  - Undo the changes to the file back to last committed revision by checking it out from the HEAD

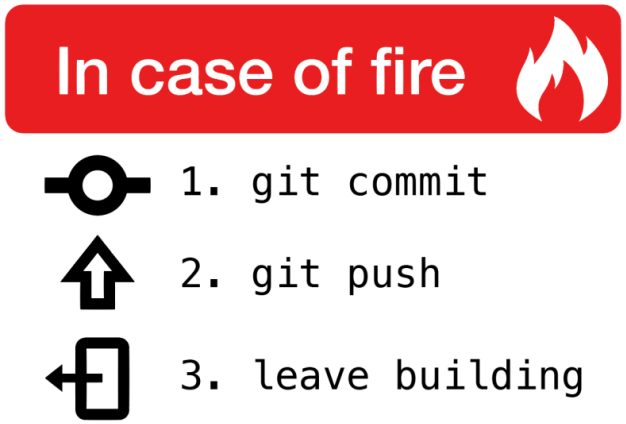
```
$ git checkout -- <filename>
```

  - Commit your changes and then redo the pull (and potentially merge the changes, if applicable)

Questions?

To summarize, add 3 commands to your daily workflow for unlimited undo and online backups of your code!

```
$ git add <files>
$ git commit -m <message>
$ git push
```



Even more information:

- Great in depth tutorials on all things git:
  - try.github.io
  - https://play.instruqt.com/public/topics/getting-started-with-git
- Software Carpentry (thanks for the images!)
  - https://swcarpentry.github.io/git-novice/01-basics/
- Contact us: research.computing@yale.edu