Project 4: Github Instructions – Draft

Carnegie Mellon University

Cyrus Daruwala, Xiong-Fei Du, Elias Roche,

Kylee Santos

April 8, 2019

Contents

Background Information

TODO

Setting up your Account

TODO

Setting Up Git Repositories (Elias)

`git init`

The `init` command creates an empty git repository – a `.git` directory containing everything you need for version control in this repository.

Command Line

To initialize a repository from the command line:

1. Navigate to desired parent directory:

- *This directory will contain the git repository.*

```
$ cd ~/cmu/76270/project4/
```

2. Create a directory for your repository:

```
$ mkdir github-instructions
```

3. Navigate to new directory:

```
$ cd github-instructions
```

4. Initialize the repository:

```
$ git init
```

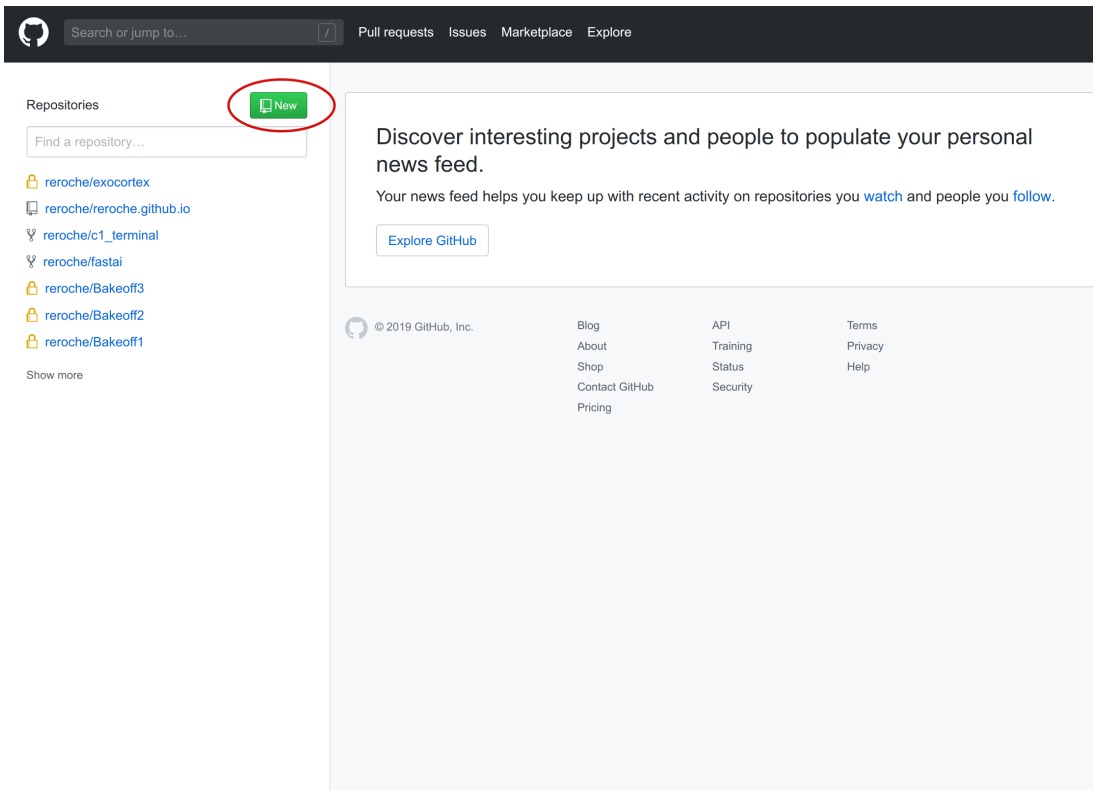
GitHub

On `github.com`, the `init` command is run automatically when a repository is created. Ensure that you are logged in before following these steps:

1. Navigate to home page:

- `http://github.com`

2. Click on the green **New** button on the top-left of the page:

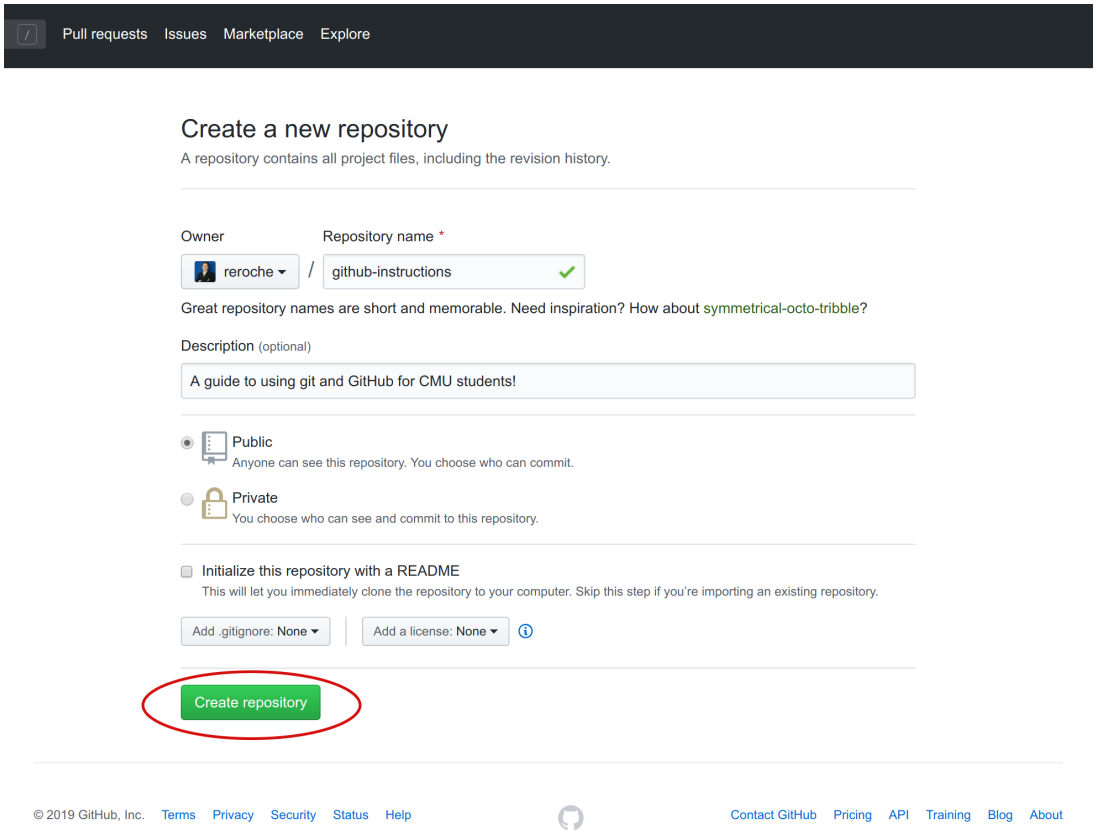


3. Enter a name for the repository:

- *The name of the repository must be unique to your user – that is, you cannot give two repositories the same name.*

4. Enter additional customizations:

- *You can include an optional description to provide more information about the purpose of the repository.*
- *Repositories can be public (others can view and use your code), or private (only you and invited users can access the repository).*
- *`README.md`, `.gitignore`, and `LICENSE` are typical files to include in a repository. They are addressed in a later section.*



5. Click on **Create Repository** to complete.

- *The page will be redirected to the empty repositories home page.*

6. Finally, set up the repository:

- *You can create a new repository on the command line by running:*

```
echo "# github-instructions" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin git@github.com:reroche/github-instructions.git
git push -u origin master
```

- *Or you can import an existing repository on your computer into this GitHub new repository by running:*

```
git remote add origin git@github.com:reroche/github-instructions.git
git push -u origin master
```

`git clone`

The `clone` command creates a copy of a of a repository into a newly created directory.

All repositories on GitHub have a unique URL, in the format:

```
https://github.com/username/repository-name.git
```

The repository for this instructions, for example, is available at `https://github.com/reroche/github-instructions`.

Command Line

To clone a repository from the command line:

1. Navigate to the desired parent directory.

```
$ cd ~/cmu/76270/project4/
```

2. Clone the repository using its GitHub URL:
    - *The new repository will be in a directory with its original name (e.g.* `github-instructions`).

```
$ git clone https://github.com/reroche/github-instructions.git
```

GitHub

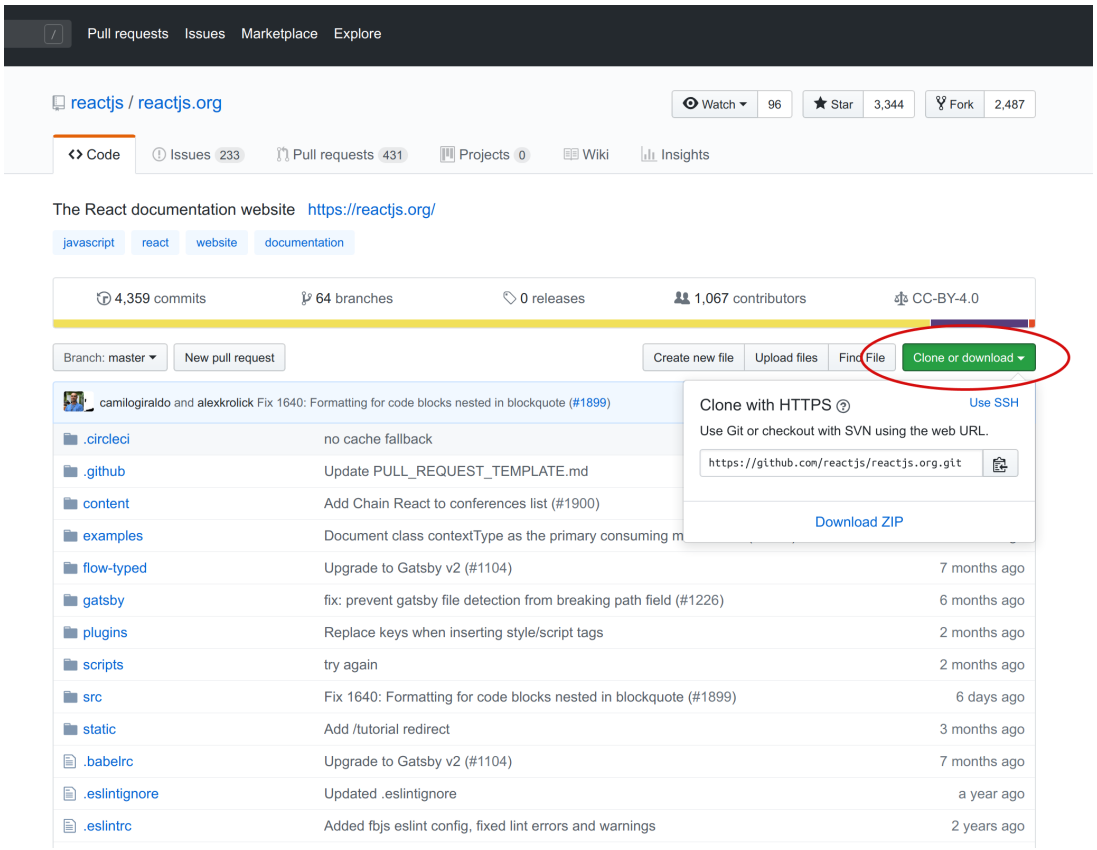
The online interface allows you to easily browse and copy repository URLs to clone via the command line.

To clone or download a repository from Github:

1. Navigate to the repository URL:

```
https://github.com/reactjs/reactjs.org
```

2. Click the green "Clone or Download" button:



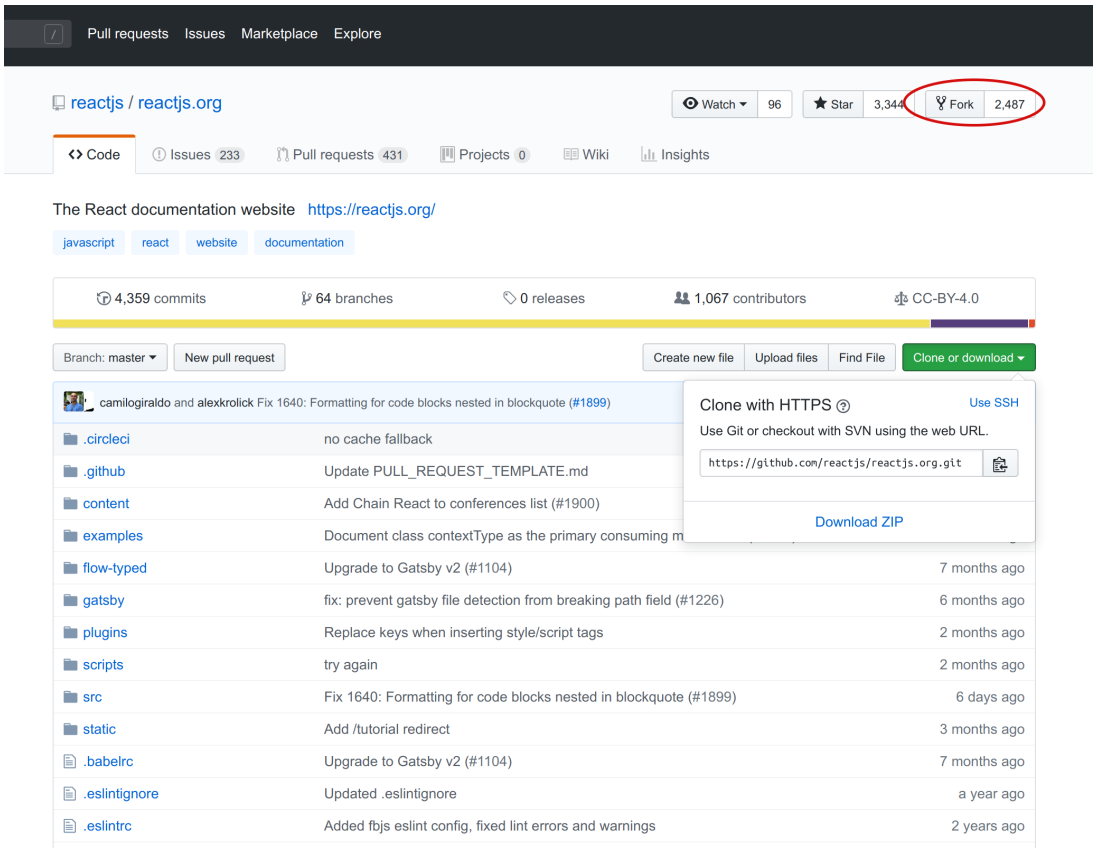
3. Copy the link and clone from the command line, or download ZIP file.

Forking on GitHub

Forking is copying a repository that belongs to another user into your repositories, granting you ownership permissions over that repository.

Forking cannot be done via the command line.

To fork a repository on Github, simply click the "Fork" button on the top right of a repository's home page.



Making Changes (Kylee)

When making changes to a repository, generally a combination of `git add`, `git commit`, and`git status` will be used. The high level procedure for making changes is the following:

1. Make edits to files in the repository.
2. Add edited files to the staging area so Git knows that you have changes there that you would like to save.
3. Check which files have been staged/unstaged.
4. Save these changes in a commit.

`git add`

The `add` command allows users to add files to the staging area (Step 2). This essentially lets Git know that you made changes to the file and you would like to save them.

However, using the `add` command does not actually save the changes yet. That functionality is done by `git commit`.

Here are a few ways to use the `git add` command: (1) Adding new or modified files individually, (2) Adding all tracked and untracked files in the working directory, (4) Adding all tracked and untracked files (new, modified, and deleted).

(1) Adding files individually > `bash > $ git add README.md draft.Rmd >` *NOTE:* If you include multiple files, they must be separated by a space.

(2) Adding all tracked and untracked files in the working directory > `bash > $ git add . >`

(3) Adding all tracked and untracked files (new, modified, and deleted) > `bash > $ git add -A >`

Tracked files are files that are included in the remote git repository. So, if you make changes to any of those files, they are already tracked.

However, if you create new files locally in the root directory, these files are untracked. Since you just created them, git does not track them since they are not yet in the remote git repository.

`git commit`

The `commit` command actually saves any changes to files you made. In conjunction with `git add`, the `commit` command will save all files that you have previously added to the

staging area.

Each commit is a snapshot, or a saved state, of your repository. Commits also require messages or captions to go along with them, usually describing the kinds of changes made.

This is beneficial if later on you are looking at all of your previous commits and want to see what changes you made in each one. Because these commit messages are so useful, there are two ways to write them.

(1) One-line commit message > `bash > $ git commit -m "Update add, commit, status sections in draft" >`

(2) Multi-line commit message > `bash > $ git commit >`

```
Update add, commit, status sections in draft.

draft.Rmd - Included formatted commandline instructions for each git command
README.md - Added sections for each group member

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Your branch is up to date with 'origin/master'.
#
# Changes to be committed:
#       modified:   README.md
#       modified:   draft.Rmd
#
~
~
```

`git status`

The `status` command allows you to display any changes you have made to the directory you are currently working in. It details which changes have been staged (using `git add`), which changes have not, and which files are not being tracked by Git.

Essentially, you will be able to see which files Git will save if you were to commit. This allows you to first, see which files you have changed, and second, confirm all the files you want/don't want to track.

Workflow

Going back to the high level procedure of how to make changes, here is an example of how to use these commands in conjunction.

Suppose I am working on a repository called `github-instructions` and I have just made changes to the `README.md` and `draft.Rmd` files.

Using `git status` gives me a summary of the files I have changed, and further `git add` can move these changes to the staging area to be committed.

Using `git status` once again will now display an updates summary displayed that the files are now in the staging area. Finally, I can save these changes with a one-line commit message.

And just like that, we have successfully saved changes to our repository!

Tools for Working Locally (Xiong-Fei)

`git stash`

The `stash` command stores away your current local changes to revert to the previous commit. This is typically done when you have made changes to a repository that you have not

committed, but want to undo.

Command Line

Suppose you already have local changes in the repository that you want to stash away. To stash changes in command line:

1. Navigate to desired directory:

```
$ cd ~/cmu/76270/project4/github-instructions/
```

2. Clear uncommitted changes:

```
$ git stash
```

Now observe that your local changes are now gone. To bring these changes back, you can run the following command:

```
$ git stash pop
```

`git log`

The `log` command allows you to see the history of commits made to your repository.

Command Line

To see the history of commits:

1. Navigate to desired directory:

```
$ cd ~/cmu/76270/project4/github-instructions/
```

2. To view commit history:

```
$ git log
```

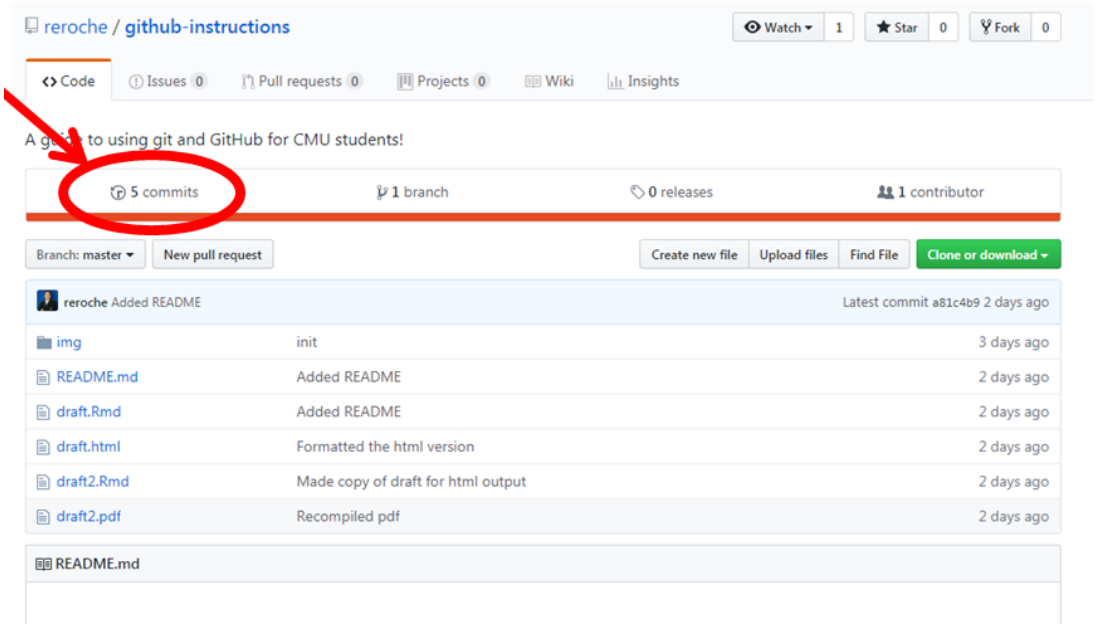
Now observe the history of commits to this repository. What you will see in your terminal is a series of commits in the following format.

Each commit has a commit hash, the user who made the commit, the time of the commit, and the commit message. The name of the branch, the commit hash, and the commit message are denoted in the above figure.

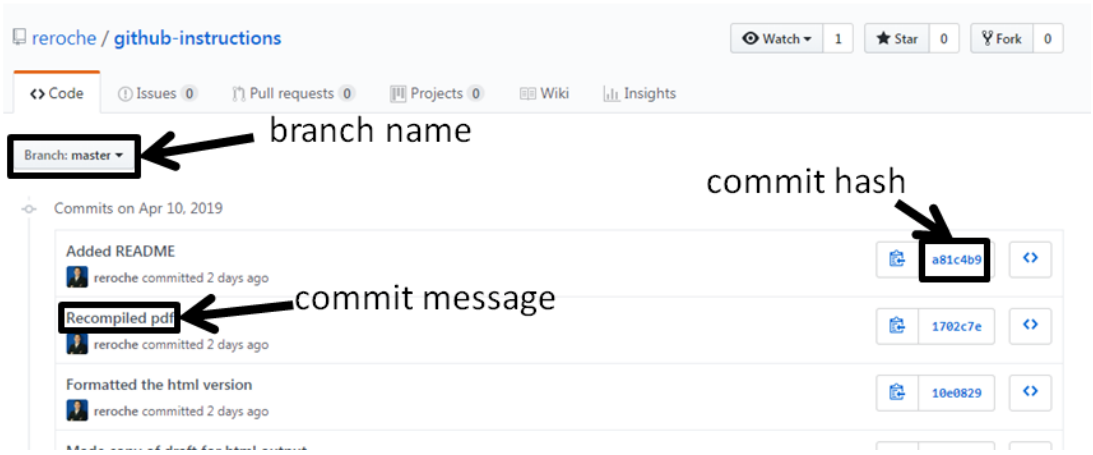
GitHub

We can also see a log of our commits in Github. This can be done as follows:

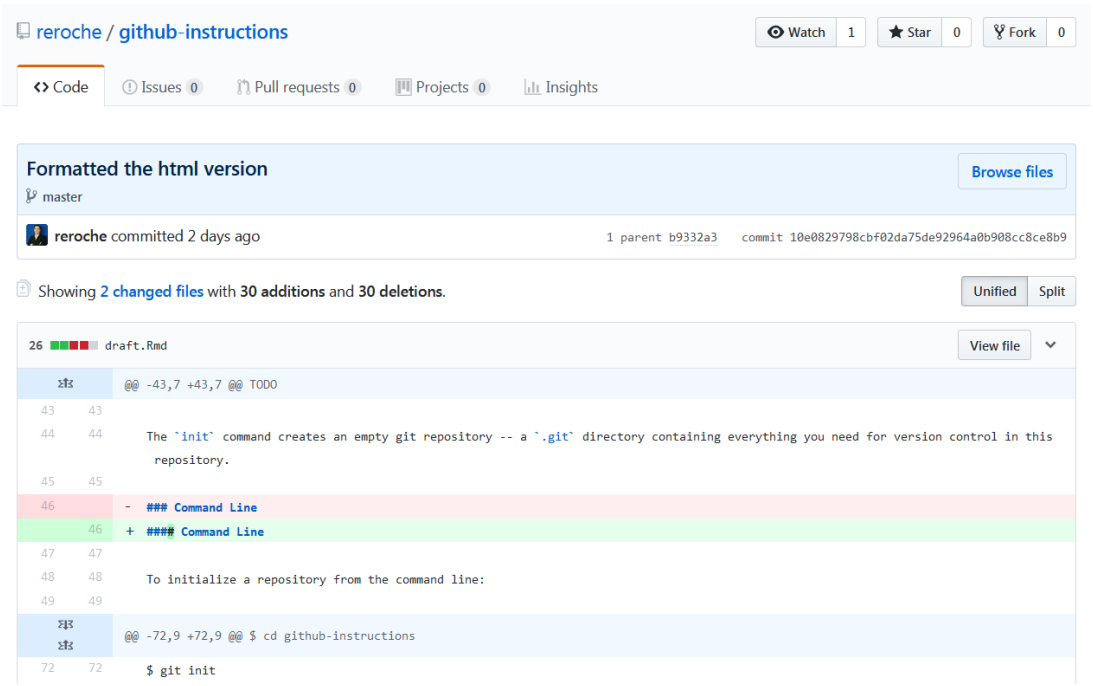
1. From the homepage of your Git repository on Github, click on the “Commits” tab:



2. Now we can see the history of commits, in the following format:



3. To view a commit in more detail, click on the commit message. Now you will see how that commit modified the files in our repository. The lines highlighted in red denotes what was removed, and lines highlighted in green show what was added:



`git diff`

The `diff` command allows you to see local changes you have made, but not yet committed, to your repository.

Command Line

To see your local changes:

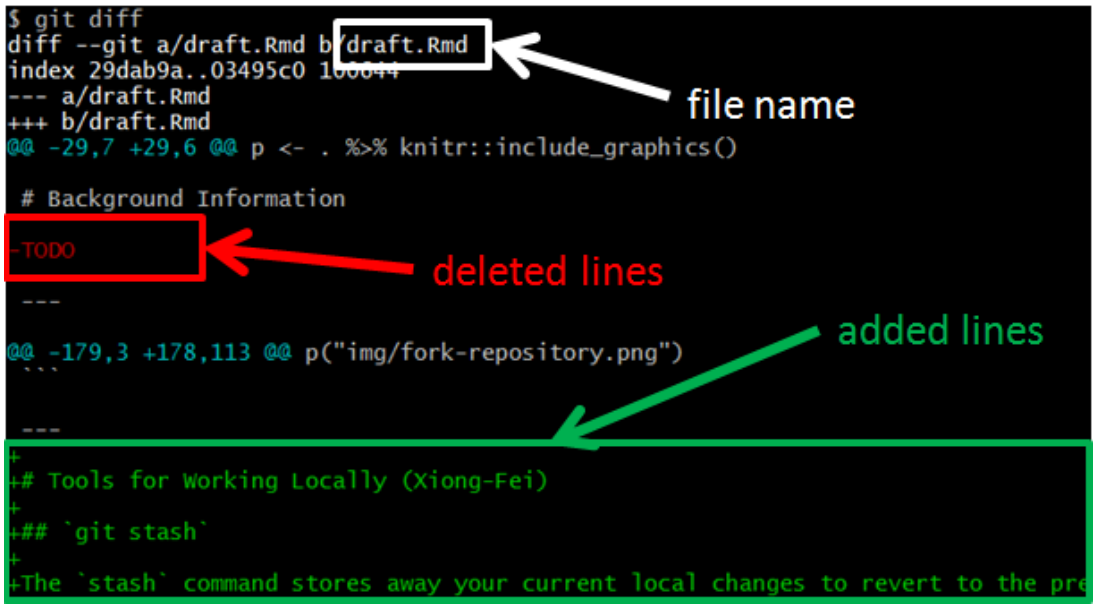
1. Navigate to desired directory:

```
$ cd ~/cmu/76270/project4/github-instructions/
```

2. To view local changes:

```
$ git diff
```

Now observe your local changes to this repository. The file name is highlighted in the following figure. Meanwhile, lines with red text show the contents of the file that were deleted while lines with green text show the added content since your last commit.



While you are in `diff`, you may hit the Enter or Return key to see more lines of differences. You may also hit Q to exit.

Syncing your local files with Github (Cyrus)

`git push`

The `git push` command is used to trasfer committed changes from a local repository to a remote repository. This allows you to make your changes available to the rest of the users of the repository.

Command Line

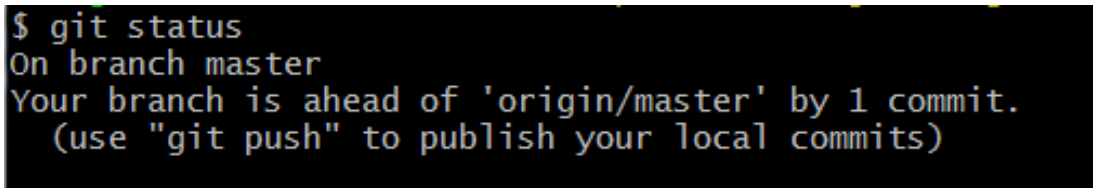
1. Navigate to desired directory:

```
$ cd ~/cmu/76270/project4/github-instructions/
```

2. Before continuing, make sure that you have committed your changes you want to push using `git commit`. To check the number of unpushed local commits:

```
$ git status
```

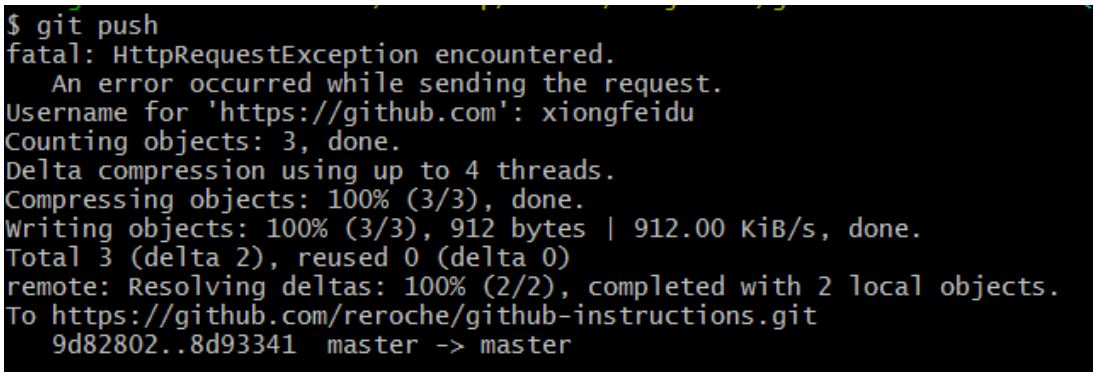
The terminal output would look as follows:



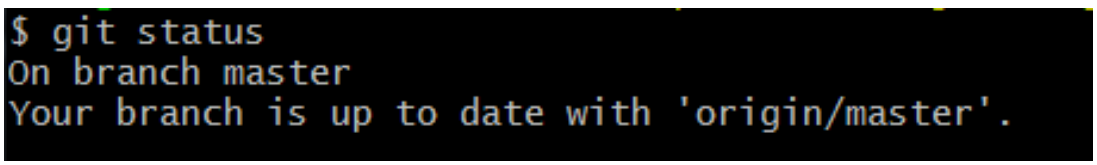
3. In the result above, it says that our branch (the local repository) is ahead of origin/master (the remote repository) by 1 commit. To push these changes:

```
$ git push
```

The terminal output would look as follows:



4. If we now check the number of unpushed local commits using `git status`, we notice that our branch is up to date (0 commits behind) with "origin/master".



`git pull`

The git pull command is used to transfer changes pushed by other users on the remote repository to the user's local repository. This allows you to view everyone else's pushed changes in your local repo.

1. Navigate to desired directory:

```
$ cd ~/cmu/76270/project4/github-instructions/
```

2. To obtain changes made by other users:

```
$ git pull
```

The resulting terminal output: