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
```

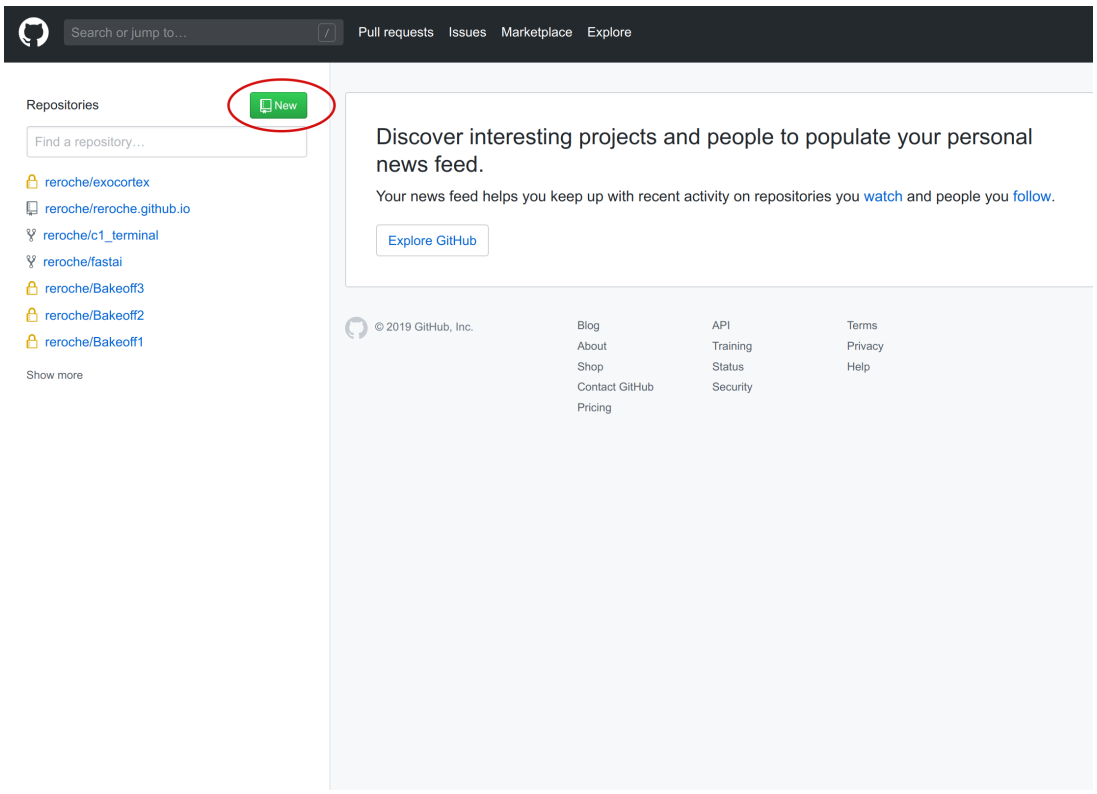
GitHub

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:

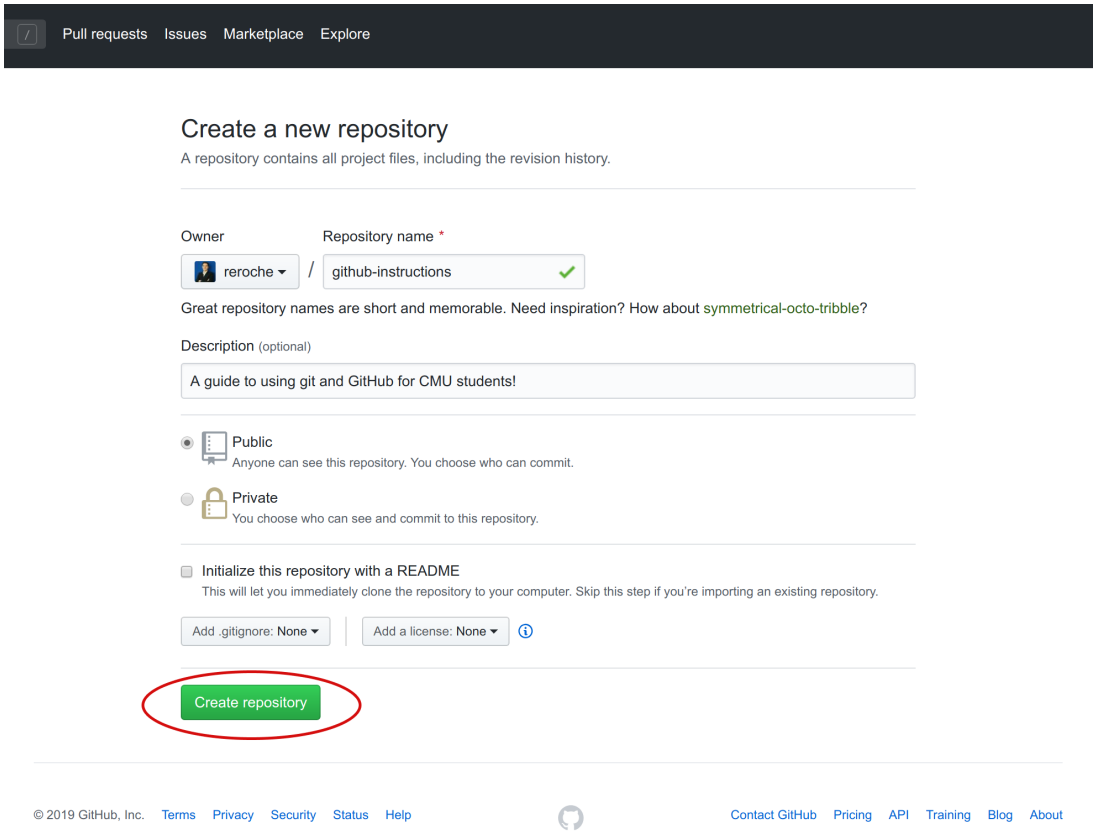


3. Enter a name for the repository:

- *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

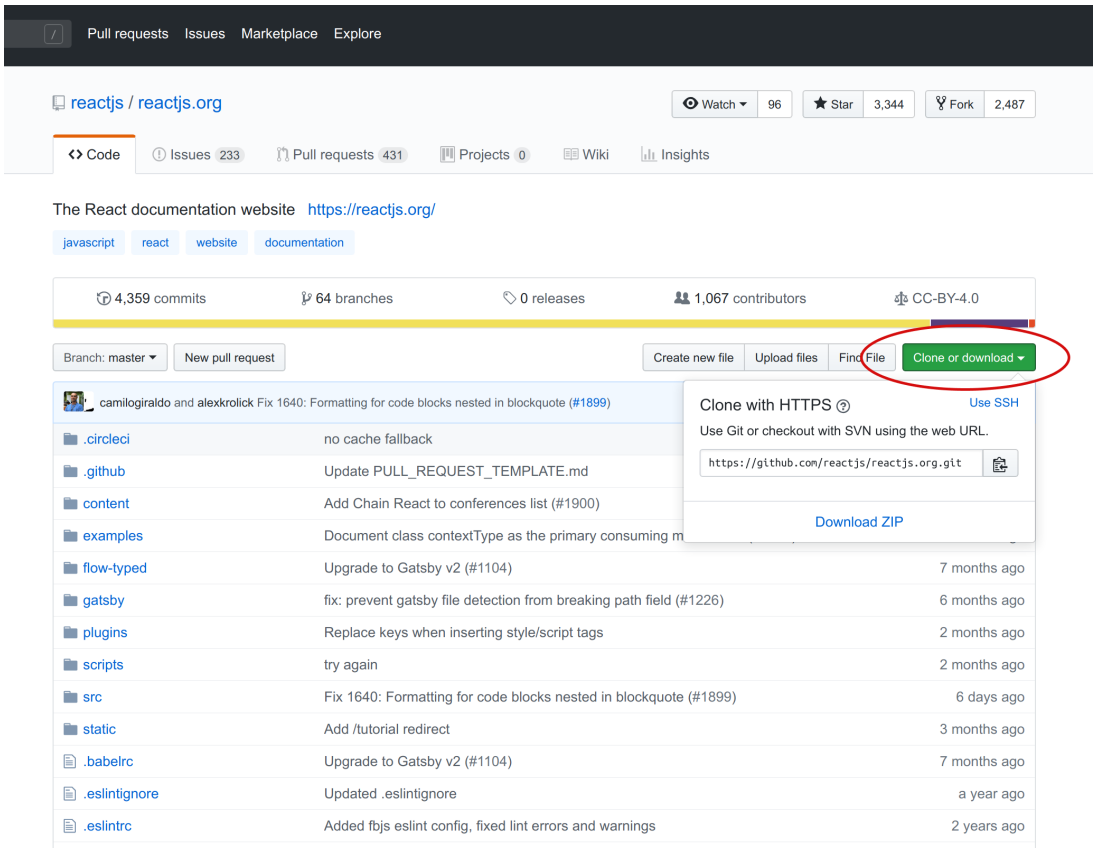
The online interface allows you to easily browse and copy repository URLs to clone via the command line.

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:



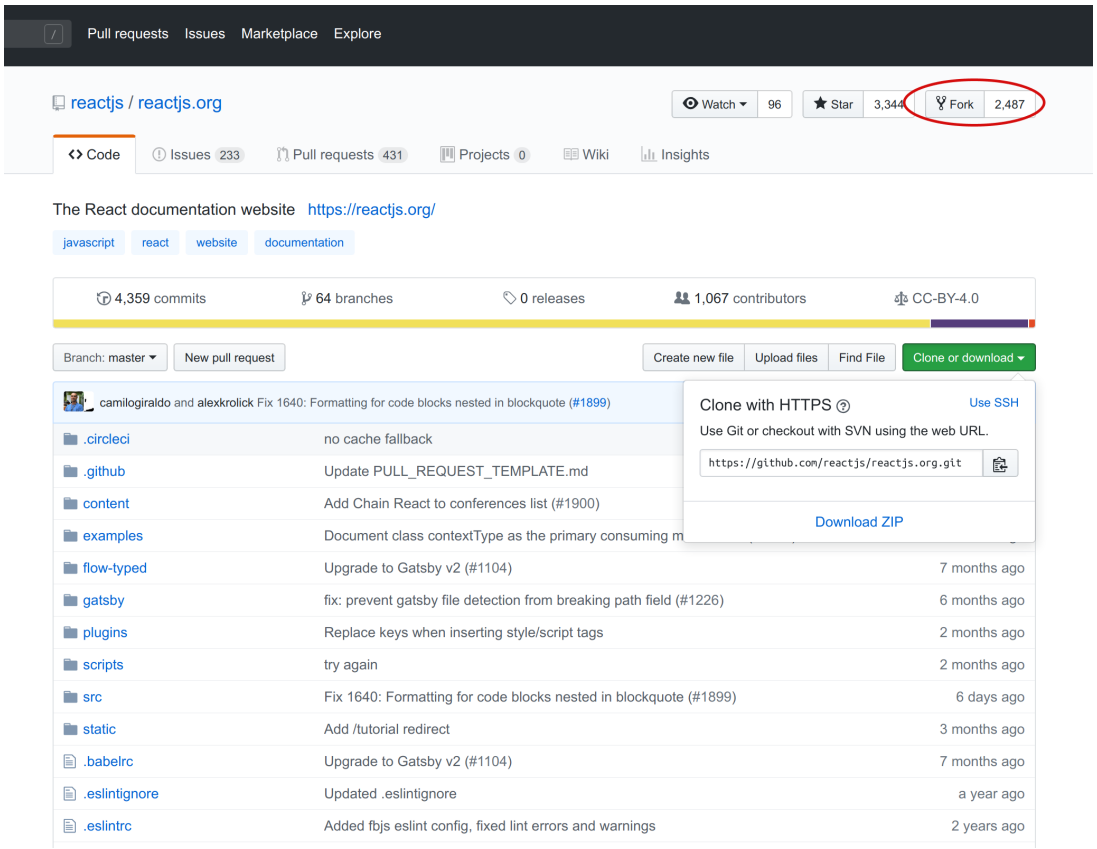
3. Copy the link and clone from the command line, or download ZIP file.

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`.

`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 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 you are looking at all of your previous commits and want to see what changes you made in each one.

`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 `Video Search Engine` with the following directory tree.

```
Kylees-MacBook-Pro:Video-Search-Engine kyleesantos$ ls
LICENSE                 package-lock.json       server.py
README copy.md          package.json            static
README.md               preprocessing           templates
gulpfile.js             requirements.txt        videos
Kylees-MacBook-Pro:Video-Search-Engine kyleesantos$
```

I made some changes in the files `preprocessing/parser.py`, `README.md`, and `requirements.txt`. I also added some videos to run my code on for testing purposes.

I can use `git status` to give me a summary of the files that I have changed.

```
Kylees-MacBook-Pro:Video-Search-Engine kyleesantos$ git status
On branch master
Your branch is up to date with 'origin/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:   README.md
        modified:   preprocessing/parser.py
        modified:   requirements.txt

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

        videos/

no changes added to commit (use "git add" and/or "git commit -a")
Kylees-MacBook-Pro:Video-Search-Engine kyleesantos$
```

I can use `git add` to add the files that I changed to the staging area.

I added `preprocessing/parser.py`, `README.md`, and `requirements.txt` but I left out the `videos` because those were only for local testing and I do not want to actually save them to Git.

If I `git status` again, I can see the files that I have added to the staging area, the ones in green being the ones that I would save on commit.

```
Kylees-MacBook-Pro:Video-Search-Engine kyleesantos$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md
        modified:   requirements.txt

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:   preprocessing/parser.py

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

        videos/

Kylees-MacBook-Pro:Video-Search-Engine kyleesantos$
```

Finally, I can save the changes using `git commit` and provide a helpful message to describe what my changes did.

```
Kylees-MacBook-Pro:Video-Search-Engine kyleesantos$ git commit
[master fbb5c08] Refactor argument parsing
 3 files changed, 5 insertions(+), 1 deletion(-)
Kylees-MacBook-Pro:Video-Search-Engine kyleesantos$
```

```
Refactor argument parsing

# 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:   preprocessing/parser.py
#       modified:   requirements.txt
#
# Untracked files:
#       videos/
#
~
~
~
~
~
```

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