
Project 4: Github Instructions – Draft

Carnegie Mellon University

Cyrus Daruwala, Xiong-Fei Du, Elias Roche,
Kylee Santos

April 8, 2019

Contents

Background Information	3
Setting up your Account	3
Setting Up Git Repositories (Elias)	3
git init	3
git clone	6
Forking on GitHub	8
Making Changes (Kylee)	8
git add	9
git commit	9
git status	9
Tools for Working Locally (Xiong-Fei)	12
git stash	12
git log	12
git diff	15

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
```

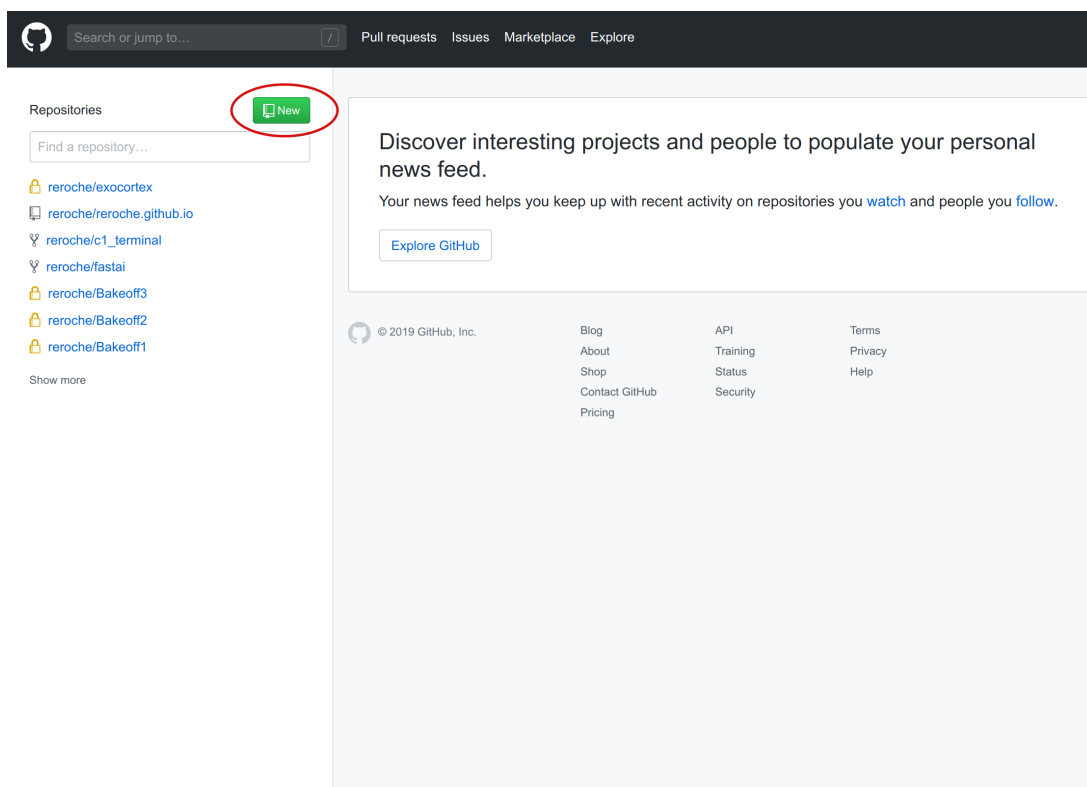
GitHub

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.*

The screenshot shows the GitHub 'Create a new repository' page. At the top, there's a navigation bar with links: Pull requests, Issues, Marketplace, and Explore. Below this, the heading 'Create a new repository' is followed by a subtext: 'A repository contains all project files, including the revision history.' The form includes a dropdown for 'Owner' (selected: reroche) and a text input for 'Repository name' (filled: github-instructions). A note says: 'Great repository names are short and memorable. Need inspiration? How about symmetrical-octo-tribble?'. There's a 'Description (optional)' text area with the text: 'A guide to using git and GitHub for CMU students!'. Below this, there are two radio buttons for visibility: 'Public' (selected) and 'Private'. A checkbox 'Initialize this repository with a README' is checked. At the bottom, there are dropdowns for 'Add .gitignore: None' and 'Add a license: None'. The 'Create repository' button is a green button, circled in red.

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 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

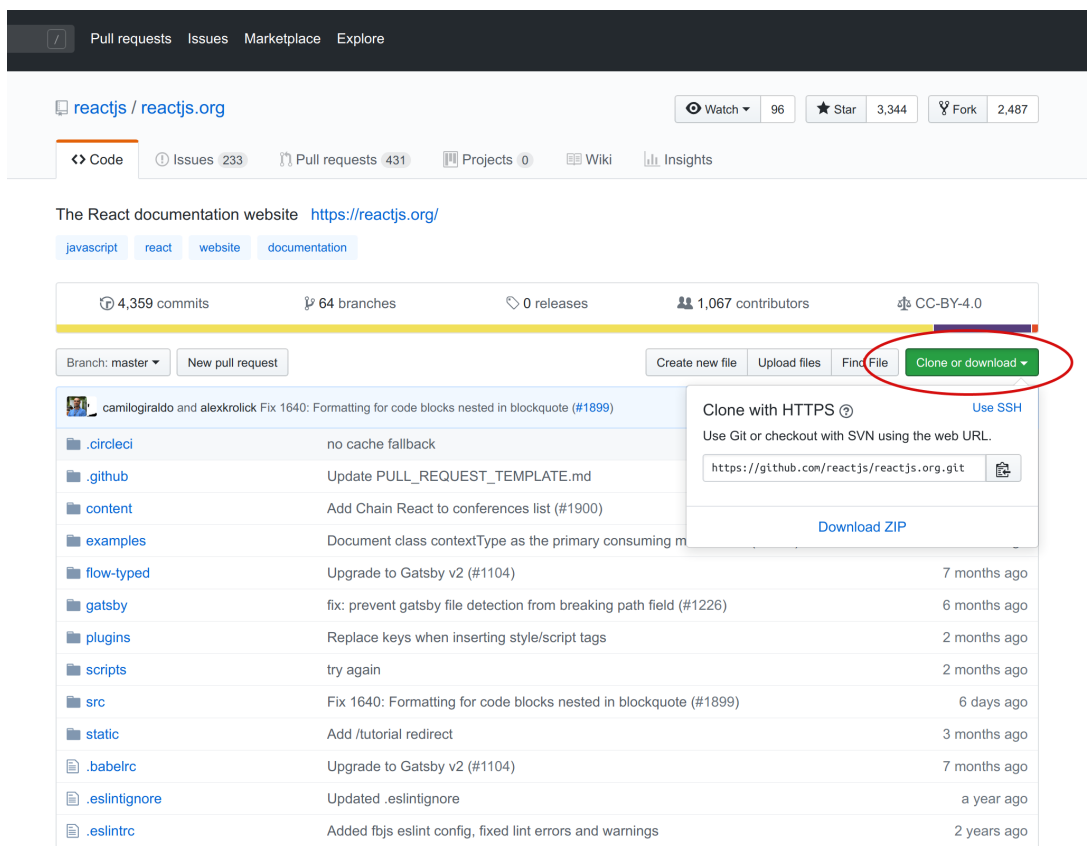
The online interface allows you to easily browse and copy repository URLs to clone via the command line.

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:



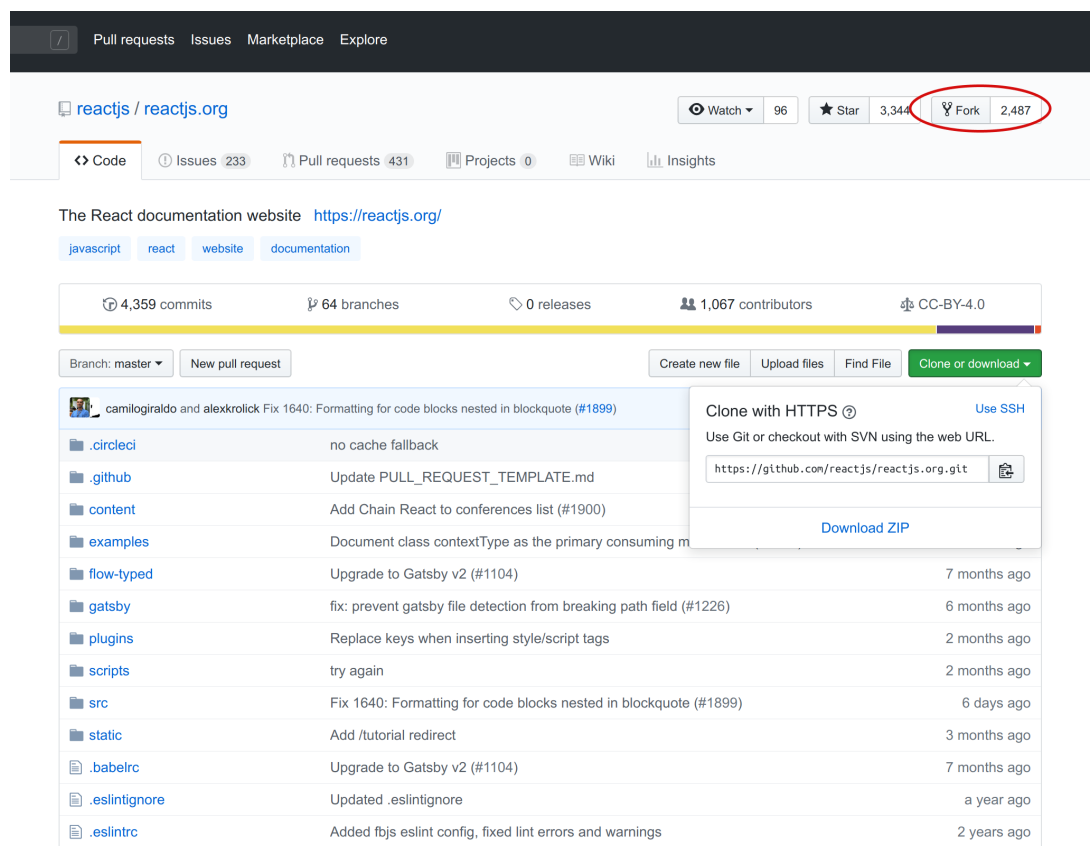
3. Copy the link and clone from the command line, or download ZIP file.

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!

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.



The image shows a terminal window with the output of the `git log` command. The output lists four commits. Annotations with arrows point to specific parts of the output:

- A green arrow points to the word `master` in the first commit's line, with the label `branch name` in green.
- A yellow arrow points to the commit hash `1702c7e55c0d2d19cb9399d184c3497f6fbafc75` in the second commit's line, with the label `commit hash` in yellow.
- A white arrow points to the commit message `Formatted the html version` in the third commit's line, with the label `commit message` in white.

```
$ git log
commit a81c4b9556f9c7f09b34201e80812d3733948c24 (HEAD -> master origin/master,
origin/HEAD)
Author: RE Roche <rroche@andrew.cmu.edu>
Date:   Wed Apr 10 15:20:46 2019 -0400

    Added README

commit 1702c7e55c0d2d19cb9399d184c3497f6fbafc75
Author: RE Roche <rroche@andrew.cmu.edu>
Date:   Wed Apr 10 15:02:06 2019 -0400

    Recompiled pdf

commit 10e0829798cbf02da75de92964a0b908cc8ce8b9
Author: RE Roche <rroche@andrew.cmu.edu>
Date:   Wed Apr 10 14:59:02 2019 -0400

    Formatted the html version

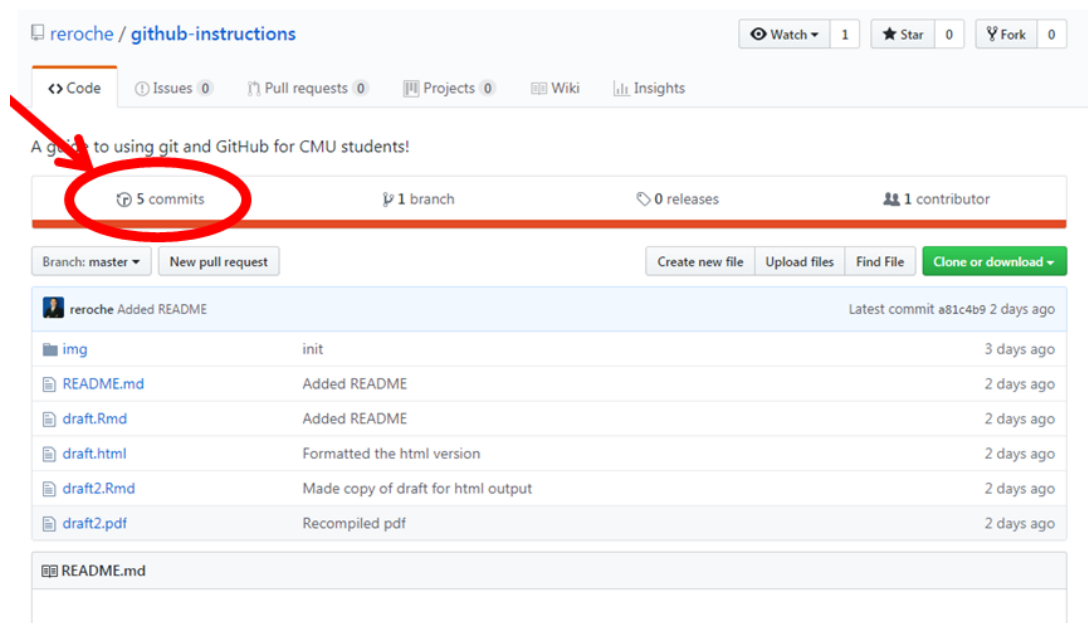
commit b9332a39c47ee6ba394a02fe4f17f04a2e1dcf9e
Author: RE Roche <rroche@andrew.cmu.edu>
Date:   Wed Apr 10 14:52:04 2019 -0400
```

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.

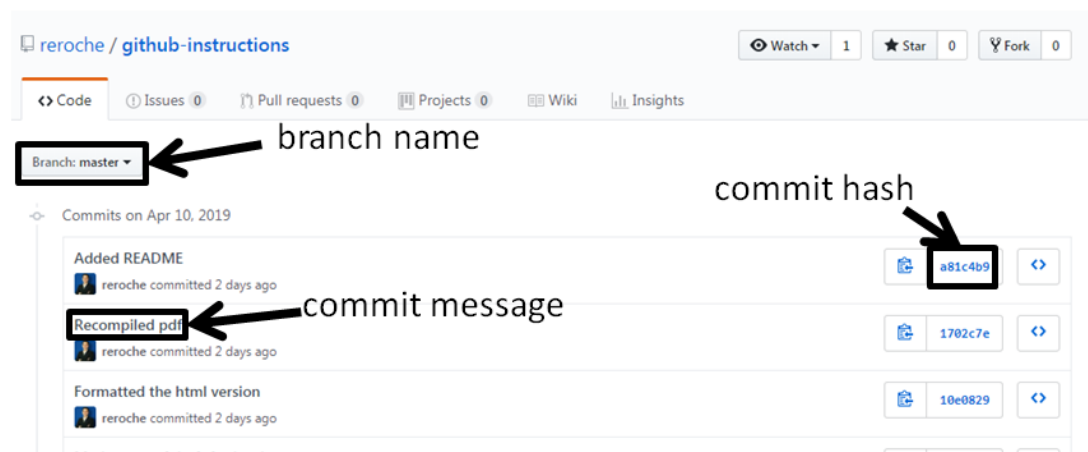
GitHub

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:

Formatted the html version

master

reroche committed 2 days ago

Showing 2 changed files with 30 additions and 30 deletions.

Unified Split

26 draft.Rmd

View file

```

@@ -43,7 +43,7 @@ TODO
43 43
44 44 The `init` command creates an empty git repository -- a `.git` directory containing everything you need for version control in this
45 45 repository.
46 46 - ### Command Line
47 47 + ### Command Line
48 48 To initialize a repository from the command line:
49 49
@@ -72,9 +72,9 @@ $ cd github-instructions
72 72 $ git init

```

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/
```

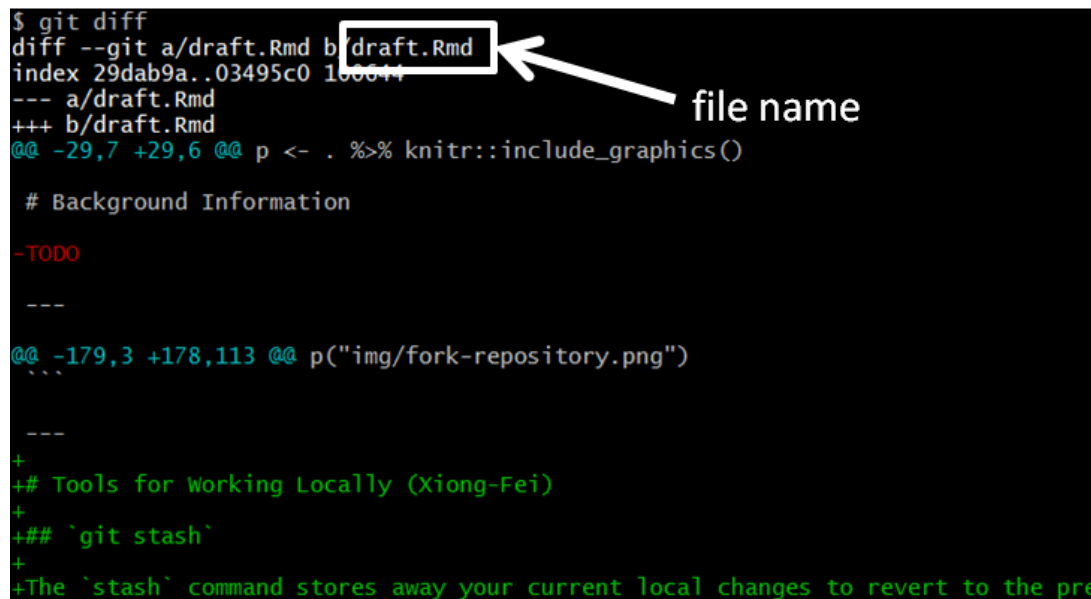
2. To view local changes:

```
$ 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.

```
$ git diff
diff --git a/draft.Rmd b/draft.Rmd
index 29dab9a..03495c0 100644
--- a/draft.Rmd
+++ b/draft.Rmd
@@ -29,7 +29,6 @@ p <- . %>% knitr::include_graphics()

# Background Information
-TODO
-
@@ -179,3 +178,113 @@ p("img/fork-repository.png")
...
-
+
+# Tools for Working Locally (Xiong-Fei)
+
+## `git stash`
+
+The `stash` command stores away your current local changes to revert to the pre
```



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.