Git Tutorial

Runpeng Li
rli3@oxy.edu
Occidental College

1 Prerequisites

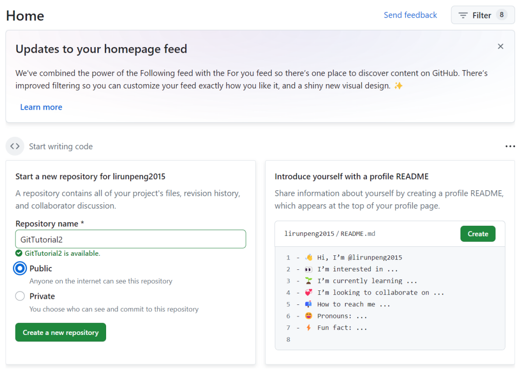
In order to follow along with this tutorial, it is suggested that you have some basic understanding of the command line. Also you will need to have git installed and a GitHub account set up.

2 What is git

Git is a robust software development tool for tracking changes in files and sharing them with collaborators. It's distributed, so every developer has a complete history of the code on their own machine.

3 To Start:

Let's kick things off by creating a new repository on GitHub. A repository serves as your Git-managed storage, tracking the history and modifications of your files. It's the hub for collaborative work with fellow developers on your project. Here's what it should resemble:



Updates to your homepage feed

We've combined the power of the Following feed with the For you feed so there's one place to discover content on GitHub. There's improved filtering so you can customize your feed exactly how you like it, and a shiny new visual design. ✨

Start a new repository for lirunpeng2015

A repository contains all of your project's files, revision history, and collaborator discussion.

Repository name *: GitTutorial2

GitTutorial2 is available.

[✓ Public] Anyone on the internet can see this repository

Private: You choose who can see and commit to this repository

Introduce yourself with a profile README

Share information about yourself by creating a profile README, which appears at the top of your profile page.

lirunpeng2015 / README.md

```
- 👋 Hi, I'm @lirunpeng2015
- 👀 I'm interested in ...
- 🌱 I'm currently learning ...
- 💞️ I'm looking to collaborate on ...
- 📫 How to reach me ...
- 😄 Pronouns: ...
- ⚡ Fun fact: ...
```

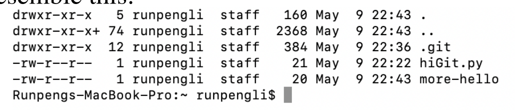
After you have created a repository, you can clone it to your computer using this command line: **git clone[URL]**

Then you can find and copy the link to the URL of your desire repository on Github by clicking the **Code** tab.

4 Creating File:

To navigate to and operate within your repository, use the 'cd' command: cd ***repository name***. Initially, your repository is empty. Let's begin by creating a Python file using the 'mkdir' command: **mkdir *your file name***. For instance, I named my file hiGit.py.

After successfully creating the file, you can use the 'ls -la' command to view it in the repository. Your output should resemble this:



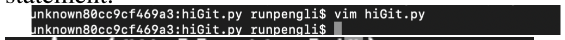
```
drwxr-xr-x   5 runpengli  staff   160 May  9 22:43 .
drwxr-xr-x+ 74 runpengli  staff  2368 May  9 22:43 ..
drwxr-xr-x  12 runpengli  staff   384 May  9 22:36 .git
-rw-r--r--   1 runpengli  staff    21 May  9 22:22 hiGit.py
-rw-r--r--   1 runpengli  staff    20 May  9 22:43 more-hello
Runpengs-MacBook-Pro:~ runpengli$ 
```

5 Editing File:

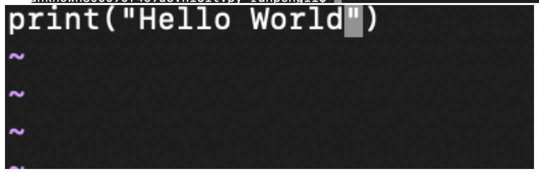
Now that we've added a Python file to our repository, we can dive into editing it and then commit our initial changes to the repository.

To edit the file, we use the 'vim' command: **vim *file name***. This will open a new window. Press the 'i' key first to insert and write code. After making changes, hit the 'escape' key and type **':wq'** (write and quit) at the bottom to save your edits.

For example, in my hiGit.py file, I added a simple print statement.



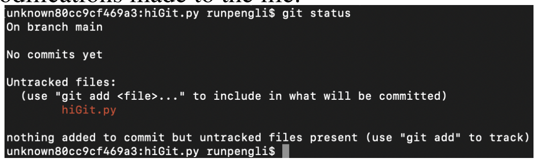
```
unknown80cc9cf469a3:hiGit.py runpengli$ vim hiGit.py
unknown80cc9cf469a3:hiGit.py runpengli$ 
```



```
print("Hello World")
~
~
~
```

6 Committing:

Let's now utilize the **'git status'** command to display the modifications made to the file.



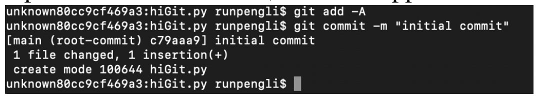
```
unknown80cc9cf469a3:hiGit.py runpengli$ git status
On branch main

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
unknown80cc9cf469a3:hiGit.py runpengli$ 
```

Upon checking, we observe that an untracked file, hiGit, has been altered.

Our subsequent action involves committing these changes. However, prior to that, we must shift our files to the staging area. The staging area, also referred to as the index, enables us to selectively choose which modifications to include in the subsequent commit. To accomplish this, we use the command: **'git add -A'** to add all files to the staging area.

Once the file is in the staging area, we can commit it to the repository using: **'git commit -m "Commit message"'**. The commit message should describe the changes being committed. For instance, I used the message: **'git commit -m "initial commit"'**.

Upon successful commit, it should appear like this:



```
unknown80cc9cf469a3:hiGit.py runpengli$ git add -A
unknown80cc9cf469a3:hiGit.py runpengli$ git commit -m "initial commit"
[main (root-commit) c79aaa9] initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 hiGit.py
unknown80cc9cf469a3:hiGit.py runpengli$ 
```

7 Branches:

branches in Git. Branches are essential for isolating development tasks such as feature enhancements, bug fixes, or experimenting with new ideas without disrupting the stability of the "main" branch. This feature allows multiple developers to work on different tasks simultaneously within the same project.

To demonstrate the use of branches, let's start by cloning an existing repository into a new one. First, create a new repository named **byeGit** and clone it locally. Before cloning the old repository into this new one, we should first move into the new repository using the command: **cd byeGit**. Check for any existing **.git**directories that could cause a conflict by using the **ls -a** command. If you find a **.git** directory, remove it with the command: **rm -rf** .git to avoid cloning errors.

Next, clone the original repository directly into this new one by using: **git clone ../GitTutorial2 ..** This action will import all contents, including the **hiGit.py** file, into **byeGit**. Proceed to edit **hiGit.py** within this new repository, perhaps using the vim command to add another print statement.

Now, before pushing any changes to your target repository, you'll need to create a temporary branch in the old repository. Navigate back to the old repository and create a new branch using: **git checkout -b [temporary branch name]**. Once on the new branch, you can return to the byeGit repository and push the changes.

At this stage, it's also a good idea to create a new permanent branch in the byeGit repository using: **git branch [new branch name]**, and then switch to it with: **git checkout [new branch name]**. You can see this new branch listed when you execute: **git branch**.

8 Merging

After making changes on your new branch, consider merging these back to the main branch to integrate your updates. Commit your changes on the new branch, then push them to the original repository using: **git push -u origin [branch name]**. To merge, switch back to the main branch **(git checkout main)**, merge the new branch **(git merge [name of branch])**, and finally, push the merged changes **(git push origin main)**

9 Deleting Branches:

.

Once merged, you can delete the obsolete branch with: **git branch -d [name of branch]**, and also ensure to remove it from the remote repository with: **git push origin –delete [name of branch]**.

10 Stashes:

Lastly, Git stashes are particularly useful for temporarily setting aside modifications that are not ready to be committed. This can be helpful when you need to switch tasks between branches. To demonstrate, make a change to **hiGit.py**, check its status with **git status**, and stash the changes with **git stash**. This will clean your working tree and safely store the changes for later re-application.

This walkthrough provides a practical guide to managing branches, merging, and stashes in Git, tailored specifically to your **hiGit.py** file and **byeGit** repository.