How to Use Git and Github

Kentaro Nakamura*

Last Updated: January 19, 2024

1 Introduction

This document is how to use git/github for your research.

2 Git for Version Control: Synopsis

Suppose that you have the following directory tree and you would like to version-control over the tree under File C.

```
FileA
├── FileB
├── FileC.....You want to control the version
│   ├── FileD
│   │   └── FileE
```

What git enables you to do is to do a control version of files (in the above case, controlling versions directories under FileC). To do this, you put (**commit**) the files into (local) **repositories**. When you get the files from stages, it is called **check-out**. To commit your files to repositories, you first need to **stage** your local files (i.e., copying your local files into the space in between your local directory and repositories). When you check-out your files from repositories, you first put your files on stage and then move it to your local working directory.

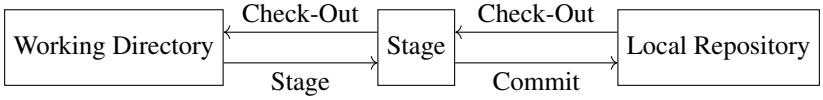

Figure 1: Stage, Commit, Repositories, and Check-Out

3 Basic Git Operation

3.1 Configuration

To set up git, you may need to set up configuration, where you set up user name, email address, your editor in use, etc. This can be done with the following commands. Below, I set VS code as a default editor for git.[1] Editor is needed when you write a commit message.

```
$ git config --global user.name "Kentaro Nakamura" //setting name
$ git config --global user.email "knakamura@g.harvard.edu" //setting email
$ git config --global core.editor "code --wait" //setting VScode as default editor
$ git config --global -e //check if editor is set successfully
$ git config --global color.ui auto //enabling color in your editor
$ git config --global -l //to see .gitconfig file
```

*Ph.D. Student, John F. Kennedy School of Government, Harvard University. E-mail: knakamura@g.harvard.edu

[1] To set VScode as a default in Mac, you may need to install 'code' command in VScode. To do so, search "Shell Command: Install 'code' command in 'PATH'" in command palette of VScode.

3.2 Initiating Version Control

You can start version control with git init command. You need to move to the working directory of interest first and then put git init.

```
$ mkdir MyProject
$ cd MyProject //changing directory to MyProject
$ pwd //checking current working directory
$ git init //initiating version control
```

3.3 Stage

You can add your files to stage (i.e., stage your files) with git add command.

```
$ git add README.txt //adding README.txt file
$ git add . //adding all files
$ git add -u //adding all the files under version control
```

If you want to make a change on a file that you already staged but before commit, you can still do it with git add FILENAME.

3.4 Commit

You can add your files on stage to your repositories (i.e., commit your files) with git commit command. You need to add commit message.

```
$ git add -u //detcting all the changed files
$ git commit //commit. editor will automatically open and you need to add the message.
$ git commit -m "making README file" //commit with one line commit message
```

3.5 Other useful commands

Here are some other basic and useful commands.

3.5.1 Check history: log

You can check history with git log command. To see history in a simplified (one-line) manner, you can use git log –oneline. To see the history of specific file, use git log –filename.

```
$ git log
$ git log --oneline //simplified view
$ git log --filename //history of filename
```

3.5.2 Check current commit, stage, and working directory: status

You can check the most recent commit (called **HEAD**), stage, and working directory with git status command.

```
$ git status
```

3.5.3 See files in stage: ls-files

You can see the files in stage with ls-files command.

```
$ git ls-files
```

3.5.4 Check difference in files: diff

You can check the difference in files (between HEAD, stage, and working directory) with git diff command. You can also compare two different commit with commit number (you only need to put the first few commit number)

```
$ git diff //between working directory and stage
$ git diff --staged //between stage and repository
$ git diff HEAD //between working directory and HEAD
$ git diff commit1 commit2 //compare commit 1 and 2 with commmit number.
$ git diff HEAD~1 HEAD~2 //compare the second last HEAD with the third last HEAD
```

3.5.5 Check commit: show

You can see how files are changed by commit with git show command.

```
$ git show
$ git show commit1 //see specific commit (commit1)
```

3.5.6 Ignore Files: .gitignore

Oftentimes, the working tree includes the files that you do not want to control the version. To enforce git to ignore the files, you need to make **.gitignore** file. For example, if you want to ignore the file 'a.pdf' and anything ended with '.log', your .gitignore file should be as follows:

```
a.pdf
*.log
```

4 Intermediate Operation: Branch

Branch is to make a multiple history of the codes. It is useful, for example, when you want to try a code for which you are not sure if it works. Traditionally, main branch for development in Git is called **master**, and after the initial commit, master branch is created, and as you commit, master branch is updated. As I mentioned in the previous section, the latest commit is called HEAD.

If you switch branch to other branches than HEAD, then as you commit, new branches are created. You can combine (**merge**) two separate branches.

4.1 Making/Checking Branch: branch

git branch commands gives you (i) what branches you currently have and (ii) where you are (with asterisk). You can name a current branch with a same command.

```
$ git branch //getting lists of branches
$ git branch NAME //name a branch
```

4.2 Switching Branches: checkout

To switch HEAD, you can use checkout command.

```
$ git checkout NAME //switch to NAME
$ git add -u //stage
$ git commit //commit (this time, creates separate branch)
```

4.3 Merging Branches: merge

You can combine two branches with git merge command. Here, merge indicates that making a new commit that reflects the changes in two different branches. Merge fails when changes in one branch overlaps changes in the other branch, which is called **conflict**. Conflict needs to be solved by editing original files. Or you can terminate merge with git merge –abort option.

```
$ git merge OTHER BRANCH's NAME
$ git merge --abort //terminate merge
```

4.4 Other useful commands

5 Use of Remote Repositories: Introduction to Github

So far, I have discussed how to make a version control in a local environment. However, you can use **remote repositories** (online repositories). The most common hosting service for repositories is **Github**.

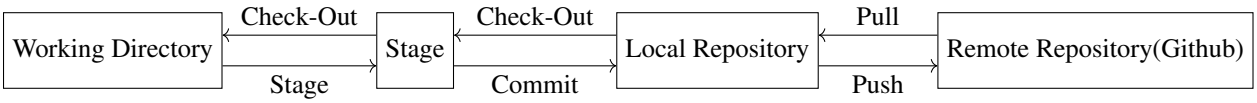

Figure 2: Remote repositories, Push, and Pull

5.1 Making Remote Repositories on Github

By putting plus botton on Github page, you can choose remote repositories. After choosing "new repositories", you fill the information on new remote repository.

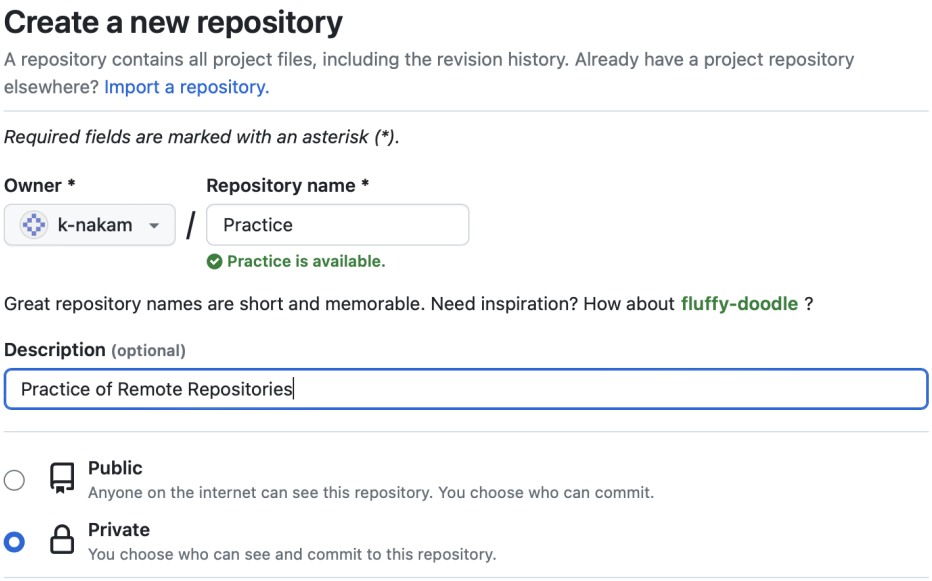

Figure 3: Remote Repositories on Github

After hitting 'create repositories', it creates **bare repository**, which does not have working tree and stages. In Github page, it shows URL links, which will be used to add remote repositories.

5.2 Adding Remote Repositories: remote

You can add your remote repositories with remote command. You can name your remote repositories.

```
$ git remote add NAME URL //add remote repository
$ git remote -v //check if remote repository is added
```

5.3 Sending to Remote Repositories: push

```
$ git push REMOTE_NAME BRANCH_NAME //e.g., git push origin master
```

5.4 Copying remote repositories to local: clone

```
$ git clone URL
$ git clone URL DIRECTORY_NAME
```

5.5 Getting remote repositories: pull

```
$ git checkout LOCAL_BRANCH //local branch name to be updated
$ git pull REMOTE_NAME BRANCH_NAME
```

6 Connect to Other Services

6.1 Overleaf

6.1.1 How to Connect Git to Overleaf

From menu command on the left, choose git and you will see the link for git clone.

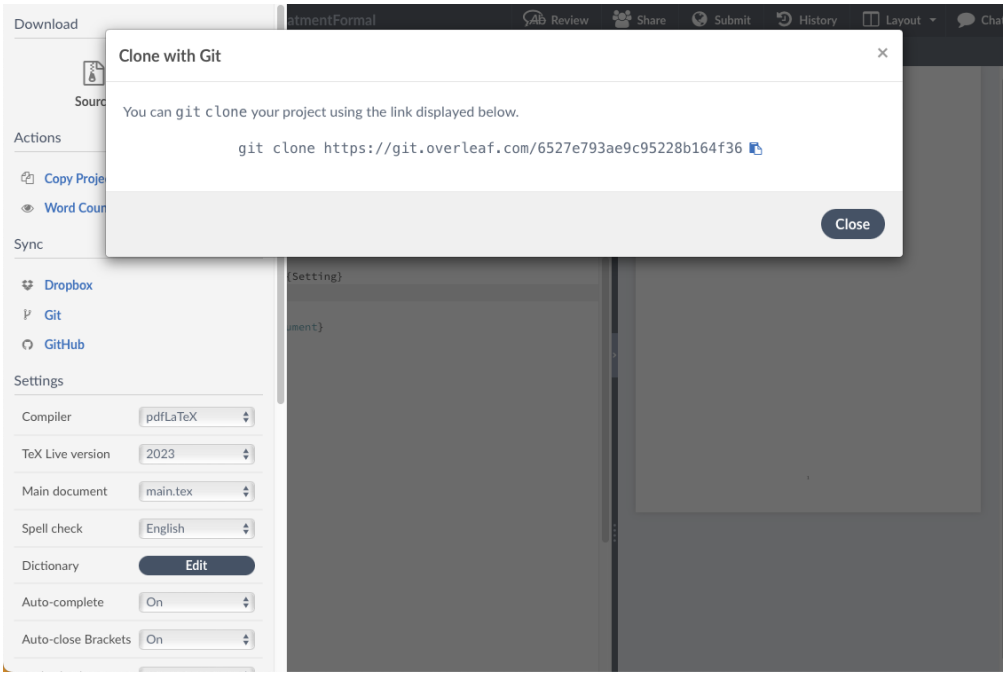

Figure 4: Get link for git from Overleaf

6.2 AWS

To be added. See https://koheikawaguchi.notion.site/Configurations-4ef443f8fabe487d8defda9a5a8b7315.

7 Useful Codes

7.1 Error: failed to push some ref to [remote repo]

In most cases, the following code fixes the error

```
$ git pull --rebase REMOTE_NAME BRANCH_NAME
```

and once you have done, you can push your local file without error.

7.2 When your .gitignore is not reflected

It is likely that you forget removing files you commit. You can do it in the following way.

```
git rm -rf --cached .
git add .
```