Git Software Carpentry Notes

- Setting up Git:

```
git config --global user.name "Suuuuuuuus"
git config --global user.email "bjzzp01@163.com"
git config --global core.autocrlf false
git config --global core.editor "code --wait"
git config --global init.defaultBranch main
```

  One can check settings by

```
git config --list
```

- Create a repository:

```
git init
```

  Change the default branch to be called `main`:

```
git checkout -b main
```

  Use the following code to show the status of our project:

```
git status
```

- To tell Git to track a file:

```
git add mars.txt
```

  as if telling Git to gather stuff before a commit. It is meaningless to `add` an empty directory to Git, but if there are uncommited files in the directory, one can `add` them all[1] by

```
git add spaceships/
```

  After that, we can let Git commit:

```
git commit -m "Notes"
```

  with some notes. Without the `-m` command, Git will launch some text editor to allow us to write a longer message. This commands is as if taking a snapshot for all stuff that has been `add` by Git. Alternatively, one may run

```
git commit -a
```

  yet it's not recommended. List all commits made to a repository in reverse chronological order:

[1] One can also use `git add -A` or `git add .` to add all files in the repository.

```
git log
```

  - Note that if the `log` is too long, Git might switch to a pager program where one can play with like a manual. To limit the number of commits one makes, simply use

    ```
    git log –1
    ```

  - Reduce the quantity of information using

    ```
    git log ––oneline
    ```

  - To display the commit history as a text-based graph:

    ```
    git log ––graph
    ```

  - One can combine `diff` and `log` by

    ```
    git log ––patch mars.txt
    ```

    It is also possible to specify a specific commit by `HEAD 3`.

- Review our changes from the last commit:

```
git diff
```

  - One can check if there is any difference between the previous commit and stuff in the staging area:

    ```
    git diff ––staged
    ```

  - One can review differences between the current file and previous commits by

    ```
    git diff HEAD~3 mars.txt
    ```

    where `~3` refers to the third-to-last commit.

  - One can also specify the ID associated with each commit by `6c498cdd333403404ad483dd94330b7a0d641999` or `6c498cd`:

    ```
    git diff 6c498cd mars.txt
    ```

- To show what changes we made at an older commit as well as the commit message, we use:

```
git show HEAD~3 mars.txt
```

- The `checkout` command allows us to go back to previous commits:

  - One can go back by one step via:

```
git checkout HEAD mars.txt
```

  - To go back further:

```
git checkout 6c498cd mars.txt
```

  - If one forgets to put the target file behind `checkout`, it might find itself in the "detached HEAD" state. This can be reverted by

```
git checkout main
```

  - Note that `git checkout` can also be used to get rid of the staged but not yet committed changes.

- We can keep track of files that we want Git to ignore by creating a file called `.gitignore` and put all filenames in. We still need to `add` and `commit` this file. Note that if any of the files in there were already being tracked, Git would continue to track them.

  - If one accidentally adds one of the ignored file

```
git add a.dat
```

    An error message will pop up. However, one can still add it by

```
git add −f a.dat
```

  - We can also always see the status of ignored files if we want:

```
git status −−ignored
```

  - We can use the `!` exclamation point operator to except a file from `.gitignore`:

```
!final.dat #except final.data
```

  - One can ignore all `.dat` files, no matter which subdirectories they are in by putting the following command in `.gitignore`:

```
∗∗/∗.dat
```

- To connect the local and the remote repository, first we need to create a repository on GitHub that has exactly the same name as our local ones. We then copy the SSH link and type:

```
git remote add origin git@github.com:Suuuuuuuus/planets.git
```

  locally. Whether this is properly done can be checked by:

```
git remote −v
```

Then, we need to setup SSH on our local PC. Simply follow instructions on this website[2]. Stuff below is only for record purpose:

```
ssh−keygen −t ed25519 −C ”bjzzp01@163.com”
Enter
kotori0803
kotori0803
ssh −T git@github.com
cat ˜/.ssh/id_ed25519.pub
```

Then, we paste the key into GitHub.

- To push the changes from our local repository to the repository on GitHub:

```
git push origin main
```

- To pull the changes from GitHub to our local repository:

```
git pull origin main
```

One can force two repositories to merge with `--allow-unrelated-histories`.

- If one has accidentally deleted some files, one may use

```
git ls−files −−deleted
```

to show all deleted files, and then

```
git checkout .
```

to retain the target files. Alternatively, it's possible to reset the local branch to what's at remote:

```
git reset −−hard origin/main
```

- The `git remote` family of commands is used to set up and alter the remotes associated with a repository[3].
  - `git remote -v` lists all the remotes that are configured.
  - `git remote add [name] [url]` is used to add a new remote.
  - `git remote remove [name]` removes a remote. Note that it doesn't affect the remote repository at all - it just removes the link to it from the local repo.
  - `git remote set-url [name] [newurl]` changes the URL that is associated with the remote.

[2] https://swcarpentry.github.io/git-novice/07-github/index.html.

[3] A remote is a copy of the repository that is hosted somewhere else, that we can push to and pull from.

  - `git remote rename [oldname] [newname]` changes the local alias by which a remote is known.

- To get the remote changes into the local repository but without merging them, one can run

```
git fetch origin main
```

  Then by running

```
git diff main origin/main
```

  one can see the changes output in the terminal.

- Miscellaneous:

  - Use the command

```
git rebase −−abort
```

    to recover the lost files[4].

  - Here is a link[5] for using Git in RStudio.

  - When trying to upload files larger than 100MB, one can refer to this link[6] for solution.

[4] I have no idea what happened before but I accidentally lost many files (quite randomly). After running this command I have everything back.

[5] https://swcarpentry.github.io/git-novice/14-supplemental-rstudio/index.html.

[6] https://blog.csdn.net/qq_42196916/article/details/105812410?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_baidulandingword~default-5-105812410-blog-105779097.pc_relevant_aa2&spm=1001.2101.3001.4242.4&utm_relevant_index=8.