**GitHub tips**

***Useful documents***

* Turing Institute Research Software Engineering course: <https://alan-turing-institute.github.io/rsd-engineeringcourse/ch02git/>
* GitHub’s own documentation: <https://docs.github.com/en/github/getting-started-with-github/quickstart>
* More documentation: <https://guides.github.com/activities/hello-world/>
* And more: <https://towardsdatascience.com/getting-started-with-git-and-github-6fcd0f2d4ac6>

***Set-up***

1. Follow the invitation to the Organisation
   1. If you don’t have an account, create one
2. Download Git (the language used by GitHub) or the GitHub Desktop client if you don’t want to use the command line
   1. Git: <https://git-scm.com/downloads>
   2. GitHub Desktop client: <https://desktop.github.com/>
3. Set up your credentials in the command line
   1. Username: <https://docs.github.com/en/github/using-git/setting-your-username-in-git>
   2. Commit email address: <https://docs.github.com/en/github/setting-up-and-managing-your-github-user-account/setting-your-commit-email-address>

Now you are able to use all of the features from the command line ☺

***Creating a repository***

A repository is basically a directory/folder where you can store a project. All code must be stored in a repository and multiple people can edit and commit (save) code to a repository.

Details of how to set one up are here: <https://docs.github.com/en/github/getting-started-with-github/create-a-repo>

In brief:

1. Create the repository on the GitHub web interface
2. ‘clone’ it (make a copy of it) onto your local computer
   1. Details of how to clone are here: <https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/cloning-a-repository>
   2. Go to the command line and navigate to the location you want the directory and execute: git clone <https://github.com/YOUR-USERNAME/YOUR-REPOSITORY>

In every repository you have a README file describing the repository.

For every repository you have the option of having a .gitignore file. This is very useful as into here you can add filenames or file endings of any files that you don’t want to be committed to the directory, e.g. program files, huge files etc.

By default, only you will have write access to the repository, unless you create it under a ‘team’. To add people to the repository to have write access, go to the repository on the webapp, navigate to settings, and then under ‘manage access’ you can invite people or teams.

***Saving your work in a repository (probably the most important part)***

1. Ideally, before you start working on some code in a repository, you should do git pull to ensure you are working with the latest version of the repository.
2. Another really helpful command is git status, this will tell you whether your branch of the repo is up to date
3. Once you have made your changes, git add <filename> will flag the file as important and something that you’d like to keep track of
   1. git add \* will identify all new files
   2. git remove <filename> similarly will flag the file as something to keep track of, this time it’ll be removed
4. now you have identified the files you are interested in, git commit -m “message” stages these files as ready to commit, i.e., ready to add the changes the repository rather than just keeping them locally
   1. the -m flag indicates that you want to write a commit message and that comes after in the quotations. It is important to always write meaningful messages, especially when working with others!
   2. If you don’t specify a message but include the -m flag, an editor will pop up and you can write it in there, but then you might get stuck in vim (I always do) so use -m “message”!
5. Now all the changes have been staged to commit, you can now push them to the master repository using git pull and hopefully everything will go smoothly

If everything doesn’t go smoothly --- **GOOGLE IT!** There is a lot of documentation online

<https://alan-turing-institute.github.io/rsd-engineeringcourse/ch02git/03Mistakes.html>

A few pointers that could help narrow down google searches:

* git reset HEAD^: \*\*used before git push\*\*\* this will rewrite history and will pretend like your previous commit never happened and there will be no log of it, nor of the reset
* git revert HEAD^: \*\*used before git push\*\*\* this will revert to the change before the commit just attempted to do
  + if you use git revert, the history will go into the git log, whereas if you do git reset, it won’t (probably better to do git revert)
* git checkout <filename>: this will overwrite your local version of the file to the file in the index

***Working together***

If you want to work with someone in a repo, use the git clone function with the repo’s address afterwards

Details are here: <https://alan-turing-institute.github.io/rsd-engineeringcourse/ch02git/05Collaboration.html>

I have created a repo in the Organisation called ‘testing\_git’ – please use this to test out functionalities of GitHub if you want, there isn’t anything important in there, so it doesn’t matter what you do to it! Everything will make much more sense once you have tested it out.

The same steps apply in terms of saving your changes those I outlined above. You just have to be careful that you aren’t overwriting other peoples’ work as that could get complicated. in general, as long as you are working on different parts of code, ideally in different files, you should be ok.

You can also clone a repo using the same account but in a different place, for example on a different device. This is a nice way of making sure your personal code is backed up and available on any device you might use.

***Forking a repository***

When you fork a repository, you make a copy of it, but you won’t make any changes to the original project.

This is helpful when you just want to use a piece of code, but you aren’t contributing to it. You can make your own changes using git add, commit, push, etc, but these won’t change the master branch.

This is probably more complicated than cloning a repository so definitely check documentation before doing it.

Details of how to fork a repo here: <https://docs.github.com/en/github/getting-started-with-github/fork-a-repo>

***Using branches***

You can create a branch of a repo which basically is a second version of your repo that you can work on in parallel.

Info on branches here: <https://alan-turing-institute.github.io/rsd-engineeringcourse/ch02git/10Branches.html>

This would be useful if you are working on changes, but you don’t know yet if they are going to be in the final version of something.