Getting started with Git and GitHub

**Pre-requisites**

Make sure you have created a free GitHub account.

Also make sure you have git installed on your machine. An installation guide can be found here: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

**Version control**

When carrying out a development project, the code base you build will come from a lot of trial and error. As part of this there will be many many revisions to the programs that you write. This is where version control can help. It can be thought of as a savings program for your project. By tracking and logging any that are made to your files, a version-control system allows you to review and restore earlier iterations. Version control takes snapshots each time you make changes to your project. You can then access these versions to compare or restore them as needed.

**Git vs GitHub**

Git and GitHub are two different things. Git is a version control system that allows you to manage and keep track of your projects history. GitHub lets you manage your Git repositories on a cloud-based hosting service. GitHub is therefore an online store for your Git repositories.

**Inputs and outputs**

Lots of code is presented in this tutorial. Remember, anything that comes after a ‘$’ is an input, code that is written by the user to perform a particular function. Anything that is presented that doesn’t start with a ‘$’ is then output. This is what the shell returns based on whatever input is used.

Inputs and outputs are shown together in the code chunks found throughout this worksheet.

**Creating a local git repository**

Any new project starts with creating a new repository (or ‘repo’) on your local machine using Git.

This is achieved by using the terminal. Navigate to the directory, or create a directory, that you want your project to reside in.

To initialise run the following command:

(cheminfo\_WS1) uos-15937:test\_repo Sam2u17$ git init

Initialized empty Git repository in /Users/Sam2u17/Desktop/test\_repo/.git/

**Adding files to the repository**

Create a file and add it to the repo.

(cheminfo\_WS1) uos-15937:test\_repo Sam2u17$ touch new\_file.txt

(cheminfo\_WS1) uos-15937:test\_repo Sam2u17$ ls

new\_file.txt

Once you've added or modified files in a folder containing a git repo, git will see that changes have been made inside the repo. However, git won't automatically keep track of the new file unless its explicitly told to.

After creating the new file, you can do the following to see which files git knows exist.

(cheminfo\_WS1) uos-15937:test\_repo Sam2u17$ git status

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

new\_file.txt

nothing added to commit but untracked files present (use "git add" to track)

Git knows a file has been made but it isn’t currently going to do anything with it.

**A quick explanation of the staging environment**

Each commitis a record of any changes to files since the last time you made a commit. Essentially, you make changes to your repo (for example, adding a file or modifying one) and then tell git to put those files into a commit.

Commits are the things that allow you to go back to the state of a project at any point.

To add a file to a commit, it needs to be added to the staging environment. To do this, you can use the git add <filename> command.

Once you've used the git add command to add all the files to the staging environment, you can then tell git to package them into a commit using the git commit command.

**Adding files to the staging environment**

Add a file to the staging environment using the git add command.

Rerunning git-status shows that the file has been added to the staging environment

(cheminfo\_WS1) uos-15937:test\_repo Sam2u17$ git add new\_file.txt

(cheminfo\_WS1) uos-15937:test\_repo Sam2u17$ git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: new\_file.txt

So we’ve not quite yet added the file to a commit, but we’re nearly there

**Committing a file**

Run the command git commit -m "Your message about the commit"

(cheminfo\_WS1) uos-15937:test\_repo Sam2u17$ git commit -m 'First commit'

[master (root-commit) 7f26cc5] First commit

Committer: Samuel Munday <Sam2u17@uos-15937.lan>

Your name and email address were configured automatically based

on your username and hostname. Please check that they are accurate.

You can suppress this message by setting them explicitly. Run the

following command and follow the instructions in your editor to edit

your configuration file:

git config --global --edit

After doing this, you may fix the identity used for this commit with:

git commit --amend --reset-author

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 new\_file.txt

A big message comes up about setting usernames, however at the moment we can ignore that and focus on the actual commit.

Each commit should be made with a descriptive name. Here we have used ‘First commit’.

Congratulations, you’ve committed a file!

**New branches**

If we want to do something new, but don’t want it to affect the main project, we can use git branches.

Branches allow you to move forwards and backwards between 'states' of a project. If you wanted to edit just a particular part of a code base, you could create a branch just for that part which wouldn’t affect the code base as a whole. Once you’re happy with it, you can then merge the changes of the branch into the master branch. Git keeps track of which commit your branch 'branched' off of, so it knows the history behind all the files

To create a new branch, do the following:

(cheminfo\_WS1) uos-15937:test\_repo Sam2u17$ git checkout -b branch\_example

Switched to a new branch 'branch\_example'

We have ‘checked out’ onto a new branch. Check that it has been created by running the following:

(cheminfo\_WS1) uos-15937:test\_repo Sam2u17$ git branch

\* branch\_example

Master

The branch name with the asterisk next to it indicates which branch you're pointed to at that given time.

If you were to switch back and make commits to the master branch, the new branch won’t see those commits until you merge them.

Lets assume that you’ve now finished editing your new branch, and want it to become part of the central code base. You would take the following step:

To switch back to the master branch:

cheminfo\_WS1) uos-15937:test\_repo Sam2u17$ git checkout master

Switched to branch 'master'

Merge the new branch with the master branch:

(cheminfo\_WS1) uos-15937:test\_repo Sam2u17$ git merge branch\_example

Updating 7f26cc5..49a1465

Fast-forward

updated\_file.txt | 0

1 file changed, 0 insertions(+), 0 deletions(-)

create mode 100644 updated\_file.txt

**Github**

If you only want to keep track of your code locally, you don't need to use GitHub. But if you want to work with a team, you can use GitHub to collaboratively modify the project's code.

To create a new repo on GitHub, log in and go to the GitHub home page. You should see a green '+ New repository' button:

A screenshot of a cell phone

Description automatically generated

After clicking the button, GitHub will ask you to name your repo and provide a brief description:

A screenshot of a cell phone

Description automatically generated

To create the repo you’ve press the ‘Create repository’ button at the bottom.

GitHub will ask if you want to create a new repo from scratch or if you want to add a repo you have created locally. In this case, since we've already created a new repo locally, we want to push that onto GitHub so follow the **'** an existing repository from the command line' section:

**Note:** Change the *<your URL>* in the first command line (git remote add origin <your URL>) to what GitHub lists as your URL.

user$ git remote add origin <*your URL>*

user$ git push -u origin master

Counting objects: 3, done.

Writing objects: 100% (3/3), 263 bytes | 0 bytes/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To <*your URL>*

\* [new branch] master -> master

Branch master set up to track remote branch master from origin.

**Pushing branches to GitHub**

Now lets push the commit in the branch you previously made to the GitHub repo. This will allow other people to see any changes that have been made. If they're approved by the repository's owner, the changes can then be merged into the master branch.

To push changes onto a new branch on GitHub, use the ‘git push origin <yourbranchname> command. GitHub will automatically create the branch for you on the remote repository:

user$ git push origin branch\_example

Counting objects: 3, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (2/2), done.

Writing objects: 100% (3/3), 313 bytes | 0 bytes/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To *<your URL>*

\* [new branch] branch\_example -> branch\_example

**Pull Requests**

A pull request (or PR) is a way to alert a repo's owners that you want to make some changes to their code. It allows them to review the code and make sure it looks good before putting your changes on the master branch.

This is what the PR page looks like before you've submitted it:

A screenshot of a cell phone

Description automatically generated

And this is what it looks like once you've submitted the PR request:

A screenshot of a cell phone

Description automatically generated

If the button entitled ‘Merge pull requests’ is green then you are good to go. If it is grey, this means there is a merge conflict. This happens when changes in one file don’t fit with changes to another file, and git is unsure which version to use.

**Merging private repos**

If you click the 'Merge pull request' button your changes will merge into a master branch.

When you're done delete the branch. This saves things getting cluttered.

You can double check that your commits were merged by clicking on the 'Commits' link on the first page of your new repo.

This shows a list of all the commits in that branch.

You can also see the hash code of the commit on the right-hand side. A hash code is a unique identifier for that specific commit. It's useful for referring to specific commits and when undoing changes. You can do this using the command: git revert <your hash code number>

**Linking changes to GitHub back to your local machine**

Your local Git wont automatically update when you merge branches (or make any changes) on GitHub. To get the most recent changes use the git pull origin master command.

user$ git pull origin master

remote: Counting objects: 1, done.

remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0

Unpacking objects: 100% (1/1), done.

From https://github.com/user/mynewrepository

\* branch master -> FETCH\_HEAD

b345d9a..5381b7c master -> origin/master

Merge made by the 'recursive' strategy.

mnelson.txt | 1 +

1 file changed, 1 insertion(+)

This shows you all the files that have changed and how they've changed.

You can then use git log to see any and all changes.

**Cloning a repository**

If you want to get a repository from GitHub onto your local machine (this could be your teams project, or an open source piece of software you want to use) you can ‘clone’that repo from GitHub to your local machine

Cloning a repository pulls down a full copy of all the repository data that GitHub has at that point in time, including all versions of every file and folder for the project. At any point, you can push your changes to the remote repository on GitHub, or pull other people's changes from GitHub.

To do this, do the following:

$ git clone https://github.com/THE-USERNAME/THE-REPOSITORY

This will create a local clone on your computer.

**Finish**

Congratulations! You now have all the tools needed to use Git and GitHub!