**Section 5.2 – Using Git**

**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 containingcl 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