An Introduction to HPC and Scientific Computing – Using Repositories and Good Coding Practice

– Practical Sessions –

Jacob Wilkins *jacob.wilkins@oerc.ox.ac.uk*
Ian Bush *ian.bush@oerc.ox.ac.uk*

Contents

1 Introduction

In these exercises we will look at using the git revision control system. Remember that `man git`, `git help` and the internet are very good resources for any issues. In particular `man giteveryday` and `man gittutorial` can be very useful. Most commands can take extra "flags" or arguments, more detailed help can be found typing `man git-[command]`.

You may also be interested in: `https://githowto.com/setup`

Remember, to log in use:
`ssh -CX [username]@oscgate.arc.ox.ac.uk`

`ssh -CX arcus-htc.ox.ac.uk`

Also, this time we will ask the SLURM scheduler to get a node where we will launch our code:
`srun --gres=gpu:1 --pty /bin/bash`

2 Planning

Before you start anything, remember that planning is important. On a piece of paper, write a plan for the following problem called "Langton's Ant":

Squares on a plane are colored variously either black or white. We arbitrarily identify one square as the "ant". The ant can travel in any of the four cardinal directions at each step it takes. The "ant" moves according to the rules below:

- At a white square, turn 90 right, flip the color of the square, move forward one unit
- At a black square, turn 90 left, flip the color of the square, move forward one unit

Recommended formats for a plan are flow-charts or pseudo-code, but anything reasonable will do. If you don't know what these are, many examples can be found online.

3 Git

Now we shall see if we can use our new knowledge to create and manage a simple repository on arcus-b.

Log onto the cluster and

1. First set up the software environment with `module load git`
2. Create a new directory using `mkdir` and change into it
3. Now create a repository by using `git init`

4. Write a program in C that displays "Hello world" on the screen. It might be useful to use the example hello.c from practical 1 for inspiration.

5. Using `git add` and `git commit` add file this to your repository

   (a) `git add` takes the files listed after it and "stages" them to be committed when you type `git commit`.
   — `git add myHello.c`

   (b) `git commit` tells git to record or update the staged files in your repository, and assign a message so that you know what you changed. Git records which files have been modified and how, so your message should be a brief summary of what your change effected.
   — `git commit -m "Fixed punctuation in hello statement"`

6. Examine the output of `git status` and `git log` and make sure you understand them

   (a) `git status` shows the current status of any changed or not-added files within your repository.
   — `git status`

   (b) `git log` prints the list of messages which you have committed.
   — `git log`

7. Modify your program to say hello to you specifically rather than to the world in general.

   (a) Use `git status` to check your change

   (b) Try using `git diff` to see what your change actually changed

8. Update the version in the repository using the appropriate commands.

9. Update your program again to not only greet you, but also to print your age and check this into the repository.

10. Write a function in C to calculate the square of an integer. Put this in a separate file to the one with the main program.

11. Add this new file to the repository.

12. Modify your main program to call this new function, and use it to print out the square of your age. Make sure you can compile and run this all correctly.

13. Check the appropriate files into your repository.

14. Finally you decide that printing out the square of your age makes you feel much too old, and you want to go back to the version where it just printed your name and your age as is. Use the appropriate git commands to go back to this version.

    (a) `git reset --hard HEAD~N` will revert back N versions

    (b) `git reset --hard <hash>` will revert to a given commit

(c) `git checkout <hash> .` will reset the files in the current directory, but may put you in a "detached head" state. We will not cover what this means, but the keen can find help online.

4 Accessing a remote Repository

Finally we shall briefly look at using git to get access to a piece of software stored on a remote repository. To do this we shall use the `git clone` command, which makes a copy of a remote repository to give you a local version. We shall look at QuEST, a library developed by Ania Brown, Tyson Jones and Jacob Wilkins which is used to simulate Quantum computers. See

`https://quest.qtechtheory.org/`

for more details.

To access QuEST all you need do is issue the command

`git clone https://github.com/QuEST-Kit/QuEST.git`

This will download the repository and create a new local one in the directory QuEST

1. change into QuEST and issue a `git status` command. Is the result what you expect?
2. Examine the `git log` and see what's there.
3. Use `git ls-files` to see all the files in the repository. Why are there so many files?
4. There is an example program using the library in this directory. Read the documentation on how to build it (Note the presence of documentation).
   - **N.B.** You may need to `module load cmake`.
5. QuEST is written in C, `cd` into the QuEST directory and have a look at the source. Note in particular how constant the style is throughout the code.