Overview

- during this tutorial we will examine how to use git from the command line
- we will also browse some of the files in the
  - linux kernel
  - and gcc
- finally we will explore a graphical visualisation of git activity
- firstly you will need to use GNU/Linux for this tutorial

Boot your computer into GNU/Linux

- if you are using Windows, reboot your computer
  - keep watching the screen and quickly choose Linux before it boots into the other default operating system
- log in to debian or mint GNU/Linux and open up a command terminal

Introduction to git on the command line

- git can be run from a gui, but here we are going to use it from the command line to expose what a git gui would be performing on our behalf
- git is a is a distributed revision control system
- useful for managing text based changes across multiple files and directories
  - distributed
  - allows files to be removed, moved
  - user can commit changes and retrieve previous changes

Introduction to git on the command line

- very useful for programming or managing documents in `groff` or `latex` format
- not so useful for `openoffice` documents
- but it could still be used
  - just not as efficient as using text

Server side: setting up your own git server

- use `mcgreg.comp.glam.ac.uk`
- so from your terminal type:
- ```
  $ hostname
  ```
- it should mention the room you are in
- now you can log into `mcgreg` by typing
- ```
  $ ssh mcgreg.comp.glam.ac.uk
  ```
- you will be prompted for a password (you should enter your linux password)

Server side: setting up your own git server

- ```
  $ cd
  $ mkdir myrepo.git
  $ cd myrepo.git
  $ git --bare init
  ```
- the above only needs to be done once

Server side: setting up your own git server

- you can log off `mcgreg`
  - do this by typing `exit` from the command line
- now check you are on your local machine by using `hostname` again

Client side

- firstly we need to define where the git repro lives
- ```
  $ mkdir -p $HOME/Sandpit/first
  $ cd $HOME/Sandpit/first
  $ git init
  $ git remote add origin username@mcgreg.comp.glam.ac.uk:/
  ```
- defines `origin` to the mcgreg url
- sets the master branch to the origin
  - for personal git repros you will rarely need to bother with branches
  - thus the `master` branch can be ignored here

Client side

```
$ mkdir -p $HOME/Sandpit/firstcopy
$ cd $HOME/Sandpit/firstcopy
$ git init
$ git add
$ touch README
$ git commit -m "My first file committed"
$ git push
```


Client side copy

- to take another copy of the files from the repository you can

```
$ mkdir -p $HOME/Sandpit/secondcopy
$ cd $HOME/Sandpit/secondcopy
$ git init
$ git clone username@mcgreg.comp.glam.ac.uk:/home/username
```

- now you could make some modifications and then

```
$ cd Sandpit/secondcopy
$ emacs README
$ git commit -m "My second commit"
$ git push
```

- which uploads changes back to the master on `mcgreg`

What changes have I made?

- can use

```
$ git diff
```

- to see the differences between your local copy and the repository

What changes have I made?

- can use:

```
$ git log
```

- to see the entire history of all changes, or:

```
$ git log version.c
```

- to see the history of all changes to the file `version.c`

Tagging in git

- you can tag a release or stage of development by:
- ```
  $ git tag version1.0 -m "version 1.0"
  ```
- to see all available tags type:
- ```
  $ git tag
  ```


To go back in time to a previous version

- ```
  $ mkdir $HOME/Sandpit/thirdcopy
  $ cd $HOME/Sandpit/thirdcopy
  $ git init
  $ git clone username@mcgreg.comp.glam.ac.uk:/home/username
  $ git checkout version1.0
  ```


To go back in time to a previous version

- or you could go back to another commit time (as seen by the `git log`)
- ```
  $ mkdir $HOME/Sandpit/fourthcopy
  $ cd $HOME/Sandpit/fourthcopy
  $ git init
  $ git clone username@mcgreg.comp.glam.ac.uk:/home/username
  $ cd myrepo.git
  $ git log
  etc etc
  ec1c3b3fa9844e3304fe24eb54e0529e356883f7
  etc etc
  $ git checkout ec1c3b3fa9844e3304fe24eb54e0529e356883f7
  Note: checking out 'ec1c3b3fa9844e3304fe24eb54e0529e35688
  
  You are in 'detached HEAD' state. You can look around, ma
  changes and commit them, and you can discard any commits
  state without impacting any branches by performing anothe
  ```


gource

- in `/usr/local/src/projects` there are copies of the linux kernel source tree and the gcc source tree
- firstly see if you can find the `ChangeLog` files in these projects

gource

- read the manual page to `gource`
- try running the program:
- ```
  $ cd /usr/local/src/projects/gcc
  $ gource -s 0.25 -highlight-all-users
  ```
- press escape to quit
- now try running it on the kernel source
- ```
  $ cd /usr/local/src/projects/linux-stable
  $ gource -s 0.25 -highlight-all-users
  ```
- press escape to quit