CSC4008 – git basics

D Greer

Repositories make a lot of sense, even if you are working individually. From a final year project point of view a big benefit is that your code is held centrally and backed up, but also you can share it with your supervisor and assessor. It also allows you to work on any machine and easily get the codebase in each case. There are other advantages too, especially feature branching where you can start a new branch and choose later whether to merge it to the main branch and of course in the future you might be part of a distributed team where there are many people working on the code base.

The basics – first you need to know there is a local installation of git and a remote repository. The local installation can be being worked with on many machines. The remote repository can be GitHub or BitBucket or (in our case) GitLab. There are several ways to use git – the best way to learn is using the command prompts. After that some people like to use a GUI version or a shell command version. We will stick to the command prompt version.

Local - Get git on your desktop

Windows: https://gitforwindows.org/

Mac: https://git-scm.com/download/mac

Linux: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Remote -Establishing a repository (repo)

Choose New Project on GitLab (via the EEECS Self Service page) and you will see a URL which points to it e.g. https://gitlab.eeecs.qub.ac.uk/dgreer/myproj.git

This would be a good time to start the README.md file. This is just a text file explaining the context of the project and to help a visitor navigate the repo.

Getting Started

Open a command prompt window, go to the folder where you keep your project files and type

```
gitinit
```

That's it you have a local tree and a remote tree. Now we need to populate them both and see how to keep them in sync.

You might already have code in the remote repo. In that case at the command prompt type

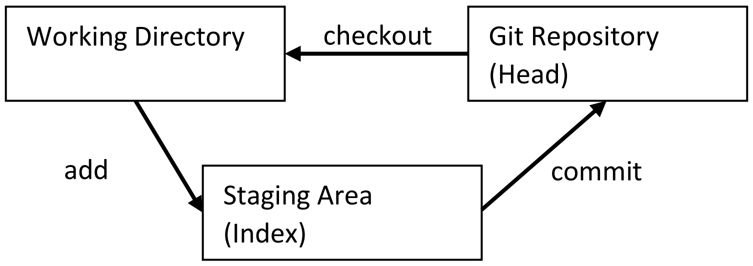
```
git clone https://gitlab.eeecs.qub.ac.uk/dgreer/myproj.git
```

replacing `dgreer` and `myproj.git` with your details.

The Change Process

Something to know early on is that local git has three trees the Working Directory, the Staging Area (also referred to as the HEAD meaning the last commit) and the git Repo. There are also three main states of files:

- *Modified* files are those in your working directory that you have not been committed to the repo.
- *Committed* changes are accepted and stored in the local repo.
- *Staged* means that you have marked a modified file as ready to be committed to the local repo.



Adding and Committing

When you want to track a file you need to tell git or in other words add it to the Index or Staging Area. To do this do

```
git add <filename>
```

To add all files

```
git add .
```

To commit to the *Local* repo

```
git commit -m "Commit message"
```

Getting things uploaded (Push)

To get everything you committed up to the remote repo.

But what if you have just done a git init and haven't specified the remote repo address. To do that

```
git remote add origin <server>
```

e.g.

```
git remote add origin
https://gitlab.eeecs.qub.ac.uk/dgreer/myproj.git
```

So to get things uploaded

```
git push origin master
```

`origin` means the repo you created in the directory you are in or the one you cloned there.
`branch` is the default name for the main branch of the remote repo tree. There can be other named branches (more on that later)

Sharing code/Team Working in GitLab

To just give access to your files click on the Gear symbol while in your project in a browser. Under "Members" you can see show to give someone whatever access you want. To share a git repository just give the other person the URL e.g. https://gitlab.eeecs.qub.ac.uk/dgreer/myproj.git

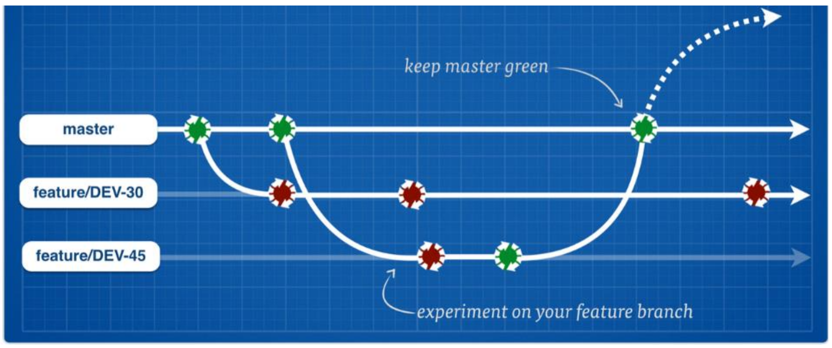
To make a copy you can use Clone. In the command window, type

```
git clone https://gitlab.eeecs.qub.ac.uk/dgreer/myproj.git
```

Note – if all you want is to store your code you can stop here

Branches

Branches are useful if you want to develop a feature separately and then decide whether to include (merge) it in your main code (more for distributed development). Create a separate branch for each feature.



(taken from https://www.atlassian.com/continuous-delivery/continuous-delivery-workflows-with-feature-branching-and-gitflow)

Suppose we want to add feature1branch- create the new branch locally using

```
git branch feature1branch
```

or if it already exists

```
git checkout -b feature1branch
```

Then work on the code for this feature. Add any files to the Index using e.g.

```
git add feature1.java
```

Then commit e.g.

```
git commit -m "feature 1 code written"
```

When happy push the feature e.g

```
git push origin feature1branch
```

On the GitLab site you should now be able to see the Master branch and the feature1 branch

Fetch, Merge and Pull

Suppose you or someone else work on another machine and set up the git repo on there. You will need to get the up-to-date code into it.

To update your local repository to match the remote one do

```
git pull
```

This is equivalent to `git fetch` and `git merge`.

To merge another branch into your active branch (e.g. master), switch to the master using

```
git checkout master
```

then merge using e.g.

```
git merge feature1branch
```

git will attempt to auto-merge changes, but it may not be able to is there are conflicts. You have to fix those first.

Mistakes - Getting back to where you where

To reset your local repo to match the remote repo (basically drop all the work since your last push) do this

```
git fetch origin
```

and

```
git reset --hard origin/master
```

If you want to remove local changes only for a certain file

```
git checkout -- <filename>
```

To get back to an old version of a file Type

```
git log --pretty=oneline
```

Establish the id of the commit you want to go back to e.g.

```
checkout 265ec7b myfile.html
```

(replace 265ec7b with the id of the first commit you made). This checkout command will also stage the old file. We still need to commit.

Suppose we don't like the old file after all. We can revert using

```
git reset HEAD myfile.html
```

This just removes the changes from staging. To revert to the latest file we need to check it out:

```
git checkout -- myfile.html
```

Tagging/Versioning

Git allows you to tag commits and this is most useful to mark a release version. To mark a commit we need its commit id**.** You can get this from the GitLab webpage or from the git log

```
git log --pretty=oneline
```

Use the first 10 characters of the id e.g.

```
git tag 1.0.0 519f881313
```

Other useful stuff

What version of git have I?

```
git –version
```

Configuring user name and email

```
git config –user.name “Des Greer”

git config –user.email des.greer@qub.ac.uk
```

(Use your own name and email.)

What is the current status of git ie is everything up to date, any modified files…?

```
git status
```