LAB 6: Setting up Version Control with Git and GitLab

Aims:

- Set up your own Git repository
    - Acquire more experience of using Git and source code version control
    - Work with GitLab
    - Set up SSH keys
- Understand the difference between the local and remote repository and the relevant commands for each
- Understand how to revert changes
- Understand branches, and how to merge changes between them
- Connect to existing repositories
- Try out GitKraken

DEMO TUTORIAL

The basic steps in setting up version control on your own computer are the following:

1. Download and install the Git version control program
2. Setup Git in your IDE
3. Test checking in and out source files
4. Setup Gitlab on the CS Gitlab server
5. Setup Gitlab in your IDE
6. Test transferring files between your local repository and the CS Gitlab server

Basic Steps using a GUI and IDE.

SETTING UP GIT with IntelliJ

- Check out this video to learn how to use GIT within IntelliJ
    - https://www.youtube.com/watch?v=mf2-MOl0VXY
    - You can also check the video from Lecture 6A.

SETTING UP A REPOSITORY USING A COMMAND LINE

In this worksheet, we will set up a local codebase, and a remote repository on the CS GitLab server.

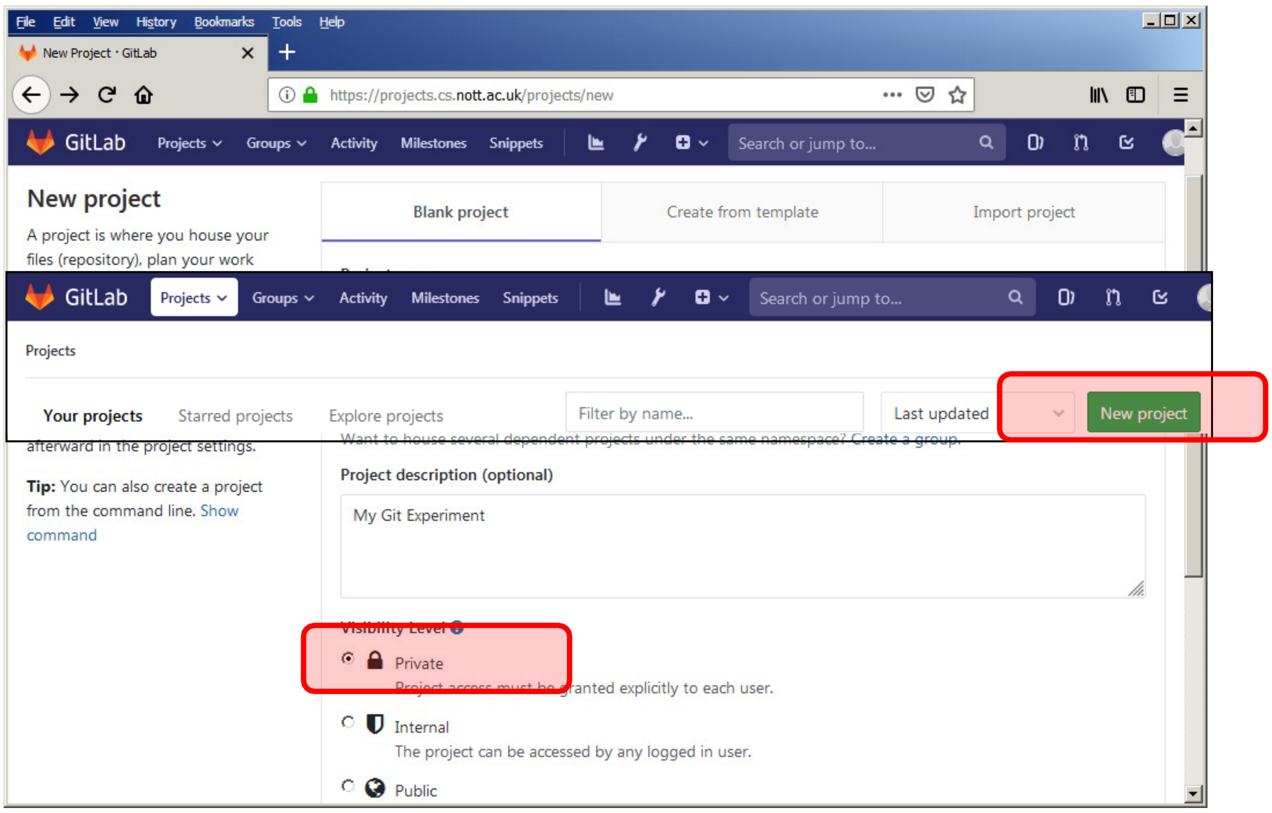
On the Computer Science standard install, you can use Cygwin to access Git from the command line. On your own machine, you can try installing http://sourceforge.net/projects/gitportable/ which when you install and run it, gives you much the same experience. You could also try using a GUI such as https://tortoisegit.org/ or GitKraken. The ideas will be the same, but the process is different. Tortoise, for example, integrates with Windows explorer, and you can check code in and out by right clicking folders or files. Some work directly with IDEs. More GUIs can be found on or from here https://git-scm.com/downloads/guis

LAB MACHINE / CITRIX NOTE: It seems Cygwin running in the lab will not keep your SSH key (in the default location) when you log off and on again. This isn't a problem for the lab, but for more serious use I would consider using GitPortable (which inherently tries to get around these issues) or changing the location of the file

Basic Steps Using the Command Line

1. First create a folder and/or navigate to it at your chosen command line prompt (Note: I used Git Bash).

2. Next create a project space on the CS GitLab server. Point a browser at https://projects.cs.nott.ac.uk. Log in, and create a new project, name it whatever you like, and make sure it is private.


File Edit View History Bookmarks Tools Help
New Project · GitLab
https://projects.cs.nott.ac.uk/projects/new
GitLab Projects Groups Activity Milestones Snippets
Search or jump to...
Projects
Your projects Starred projects Explore projects
Filter by name...
Last updated
New project
Blank project Create from template Import project

New project

A project is where you house your files (repository), plan your work

afterward in the project settings.

Tip: You can also create a project from the command line. Show command

Want to house several dependent projects under the same namespace? Create a group.

Project description (optional)

My Git Experiment

Visibility Level

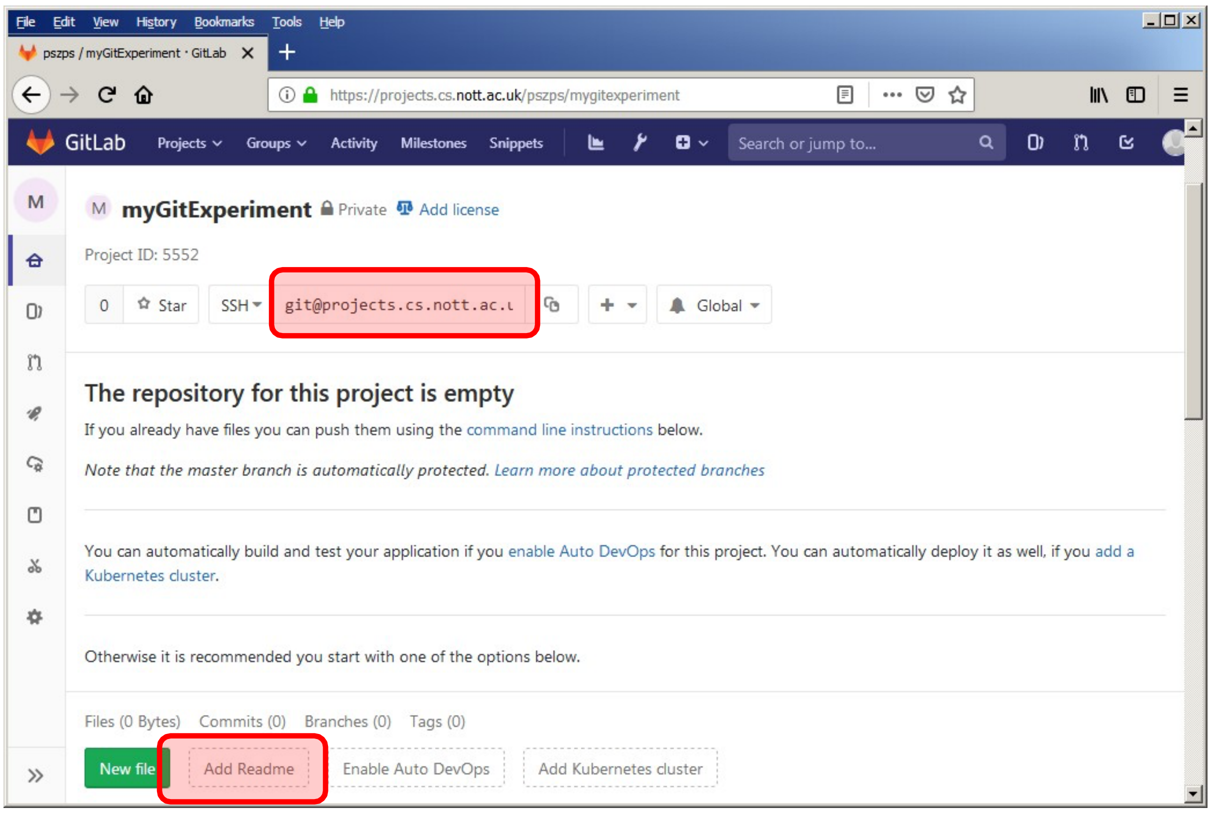
[✓ Private] Project access must be granted explicitly to each user.

Internal The project can be accessed by any logged in user.

Public

You will then be presented with a new page. The address at the top of the page shows the location of the repository. You can use this to point the Git command line client or your GUI tool of choice at the repository.

3. Add a Readme file from GitLab by clicking the "Add Readme" link.



https://projects.cs.nott.ac.uk/pszps/mygitexperiment

myGitExperiment Private Add license

Project ID: 5552

git@projects.cs.nott.ac.u

The repository for this project is empty

If you already have files you can push them using the command line instructions below.

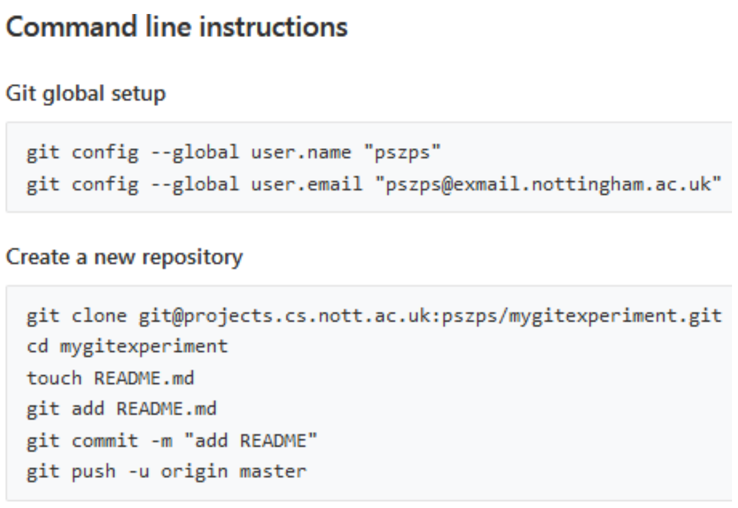
Note that the master branch is automatically protected. Learn more about protected branches

You can automatically build and test your application if you enable Auto DevOps for this project. You can automatically deploy it as well, if you add a Kubernetes cluster.

Otherwise it is recommended you start with one of the options below.

Files (0 Bytes) Commits (0) Branches (0) Tags (0)

4. Next you need to enter the 'Git global setup' details into the **git command line**, using git config. The GitLab page, (personalized for you) shows an example of how to do this.



Command line instructions

Git global setup

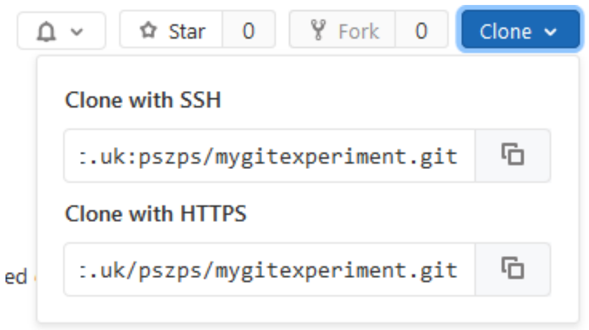
```
git config --global user.name "pszps"
git config --global user.email "pszps@exmail.nottingham.ac.uk"
```

Create a new repository

```
git clone git@projects.cs.nott.ac.uk:pszps/mygitexperiment.git
cd mygitexperiment
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

You can now follow the clone instructions shown by GitLab for "creating a new repository". The first line will set up your local folder to mirror the remote repository you have created. It should create the README.md file you just entered in your local repository.

If you get an error message when you try to clone the repository, try out an alternative way. Rather than cloning it via ssh, clone it via https. For the latter, as an additional step, you will be asked to provide your credentials.



Clone with SSH

```
:.uk:pszps/mygitexperiment.git
```

Clone with HTTPS

```
:.uk/pszps/mygitexperiment.git
```

Before you can commit remotely (push), you need to make sure that you (1) have an SSH key set up and (2) GitLab knows about it before you can push projects remotely. GitLab provide instructions. Type "ssh" in the "Search or jump to ..." field and choose "Settings/SSH Keys"; then follow the instructions.

If you get stuck, more information about SSH can be found here:

- https://docs.gitlab.com/ce/ssh/README.html
- https://projects.cs.nott.ac.uk/help/ssh/README.html

5. Now edit the README.md file locally with your favorite text editor and save it.

6. Then, at the **git command line** move to the local repository folder and type:
   - git add -A [filename, or -A . for everything]
   - git commit -m "Your message here"

7. Now look at the file again on GitLab. Can you see your changes? No! Because we have only committed to the local repository. Remember this:

8. We only added the file and committed to the local repository. Now we need to push it to the server!

   - git push -u origin master

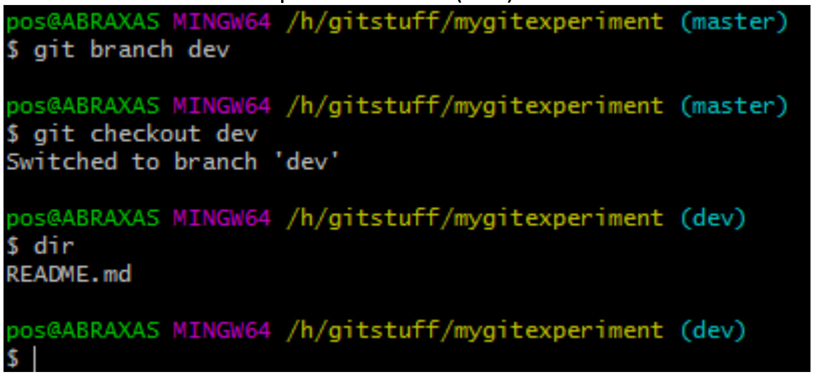
NOW check GitLab. Changes should be visible and logged!

TIP: You can use the git push -u switch to get git to save the parameters, so next time we just need to do git push

To retrieve any changes from the remote repository we can use:
$ git pull origin master

BRANCHES

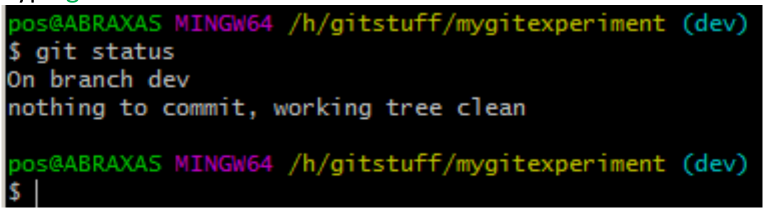
Working in branches is like using "Save as..." to copy your project to a different location, to work on it. Let's create a development branch (dev) from our local master:



```
pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (master)
$ git branch dev

pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (master)
$ git checkout dev
Switched to branch 'dev'

pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (dev)
$ dir
README.md

pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (dev)
$ |
```

Note how we change to the dev current branch using checkout. This is like changing to the new folder.

Type git status to see where we are.

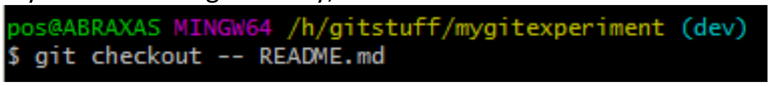


```
pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (dev)
$ git status
On branch dev
nothing to commit, working tree clean

pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (dev)
$ |
```

Let's now add a further line to our README.md - whatever and where ever you like. Then do a git add and a git commit.

ROLLING BACK CHANGES

If you make changes locally, and then want them rolled back to the latest committed version, use:



```
pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (dev)
$ git checkout -- README.md
```

You can see earlier versions using git checkout [commit number, e.g. 1db2732]

You can fetch an earlier committed file using git checkout filename

HEAD is a useful keyword here. It is the version at the tip of the current branch, e.g.:

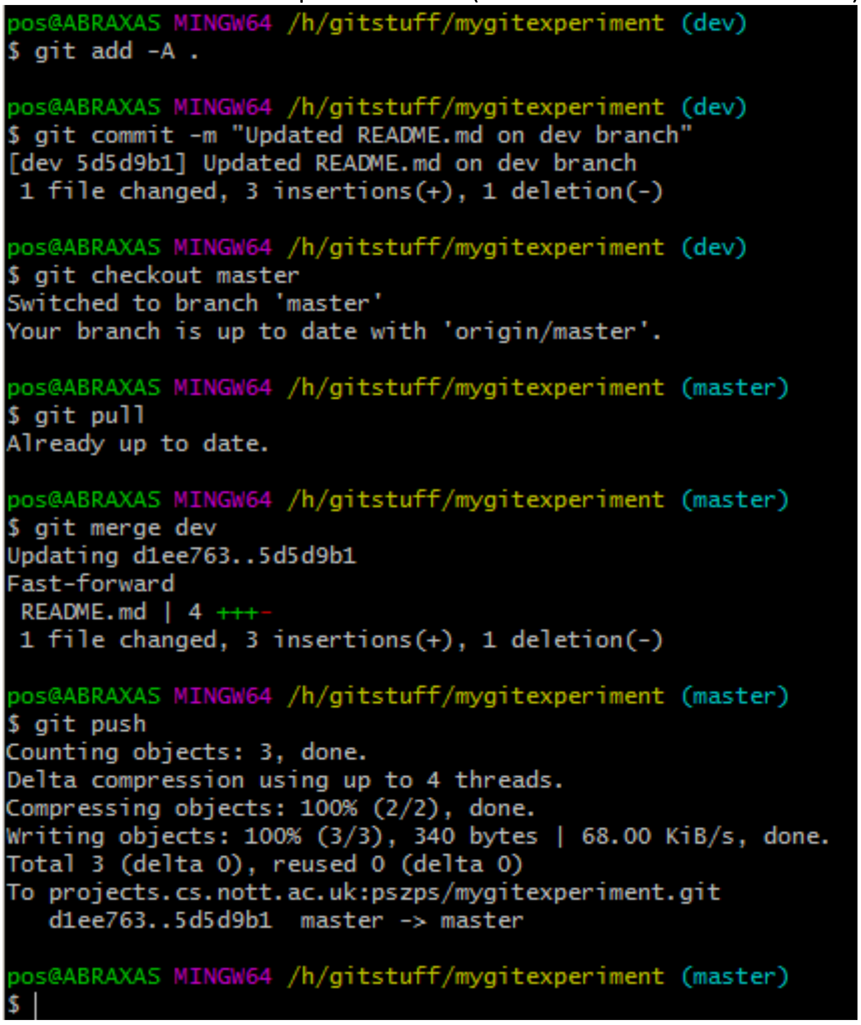
- git checkout HEAD Main.java

checks out the most recent version of this file.

Look up git revert in the internet to see what it does

MERGING

When we are happy with our committed changes, we want to merge them back into the master code set. There's a few options for this (see the workflows section later). One typical way is:

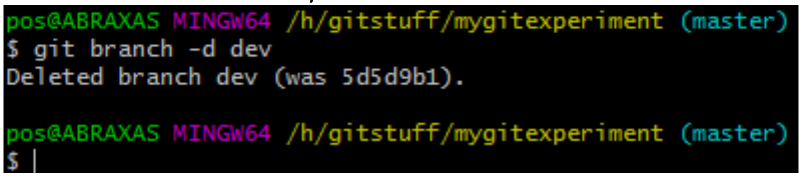


```
pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (dev)
$ git add -A .

pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (dev)
$ git commit -m "Updated README.md on dev branch"
[dev 5d5d9b1] Updated README.md on dev branch
 1 file changed, 3 insertions(+), 1 deletion(-)

pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (dev)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (master)
$ git pull
Already up to date.

pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (master)
$ git merge dev
Updating d1ee763..5d5d9b1
Fast-forward
 README.md | 4 +++-
 1 file changed, 3 insertions(+), 1 deletion(-)

pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (master)
$ git push
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 340 bytes | 68.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To projects.cs.nott.ac.uk:pszps/mygitexperiment.git
   d1ee763..5d5d9b1  master -> master

pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (master)
$ |
```

That is: switch to the master branch, pull any changes made to this branch from the server, merge your 'dev' branch into the master, and then push the merged result back to the server when you've checked there are no errors.

- Note: it is good practice to push your local branches onto the remote repository so others can see your code and make changes
- See the workflow section for a link to some more thorough examples.

To delete a branch when you have finished with it:



```
pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (master)
$ git branch -d dev
Deleted branch dev (was 5d5d9b1).

pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (master)
$ |
```

IGNORING FILES

Up to now we've talked about checking in source files or text files. There are a whole list of file types we do NOT want to check in to git. For example, we don't want to check in .class files, or binary files in general. These get changed every time the project is built, and it doesn't make sense to check them using versioning - we version the source files anyway.

Also, you do not want to check in project or settings files. These tend to be specific to you as a user. If someone else checks them out, it may mess with their own IDE setup.

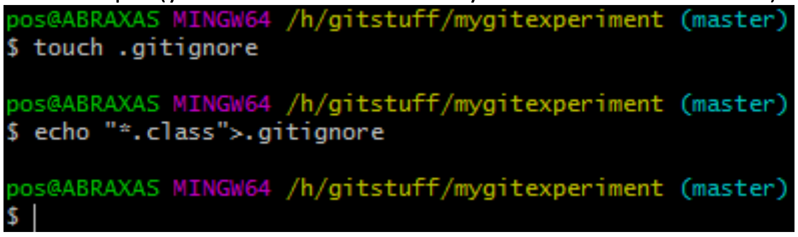
To tell git to ignore files:

Create a .gitignore file. You can create this in the project root folder, or you can set a default for all projects in git (see https://git-scm.com/docs/gitignore)

The file is simply a list of expressions to match for files to ignore, e.g.

- bin/*
- *.class

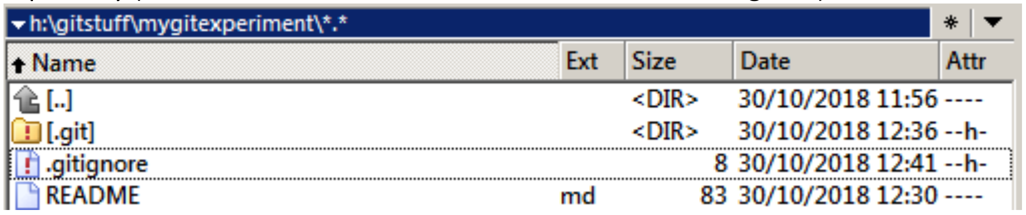
For example (you can also do this with your text editor of choice):



```
pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (master)
$ touch .gitignore

pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (master)
$ echo "*.class">.gitignore

pos@ABRAXAS MINGW64 /h/gitstuff/mygitexperiment (master)
$ |
```

Put each expression on a new line. When you next do a git add, you will see that git just ignores the files listed.

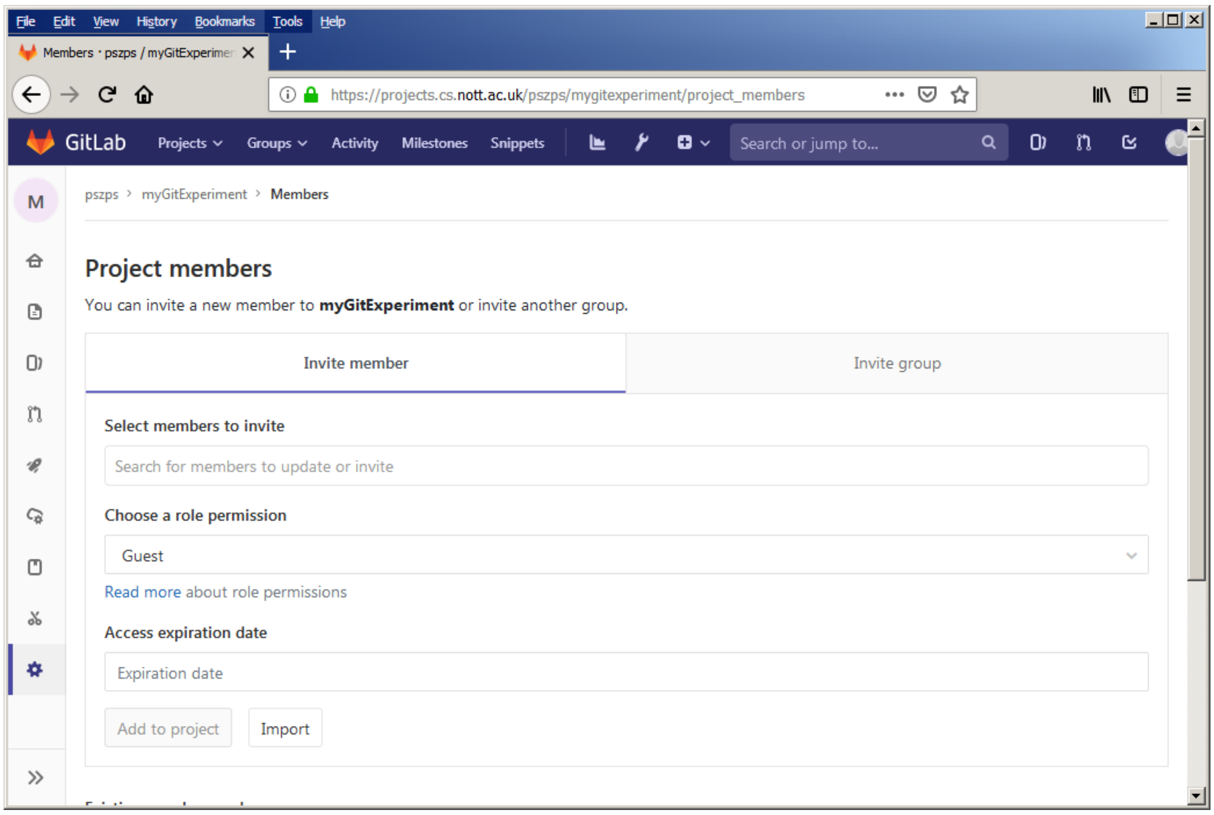
If you used the **git command line** check that a new (hidden) file has been created in the local repository (that contains the information about which files to ignore).



```
h:\gitstuff\mygitexperiment\*.*
Name                  Ext   Size   Date              Attr
[..]                        <DIR>  30/10/2018 11:56  ----
[.git]                      <DIR>  30/10/2018 12:36  --h-
.gitignore                     8   30/10/2018 12:41  --h-
README                md      83   30/10/2018 12:30  ----
```

ADDING PEOPLE TO A PROJECT

You can add people to a project to see your code via GitLab. They will then need to have their own SSH set up on their own GitLab account and GitLab will take care of the rest. Do NOT add other people's SSH keys to your account.


File Edit View History Bookmarks Tools Help
Members · pszps / myGitExperime
https://projects.cs.nott.ac.uk/pszps/mygitexperiment/project_members
GitLab Projects Groups Activity Milestones Snippets
Search or jump to...
pszps › myGitExperiment › Members

Project members

You can invite a new member to **myGitExperiment** or invite another group.

Invite member | Invite group

Select members to invite

Search for members to update or invite

Choose a role permission

Guest

Read more about role permissions

Access expiration date

Expiration date

Add to project | Import

'Add members' is on the settings page for your project. Choose the correct permissions settings! See here. https://docs.gitlab.com/ee/user/permissions.html

COMMON GIT WORKFLOWS

There are different ways you can organise your use of git as a team. Have a look at this site for some workflow ideas:

https://www.atlassian.com/git/tutorials/comparing-workflows

The workflow we used above was (nearly) a feature branch workflows. The idea is to pick a way of working that works for your team, and stick to it.

OTHER COMMANDS, AND RESOURCES

- git status - view current status
- git log - view logs
- git diff - see differences between files/commits git reset – remove staged files
- git branch - see current branch
- git rm - remove a file locally, and stage a file for removal at the next commit
- git help

See also:
Online simple introduction

- http://try.github.io/

Workflows - ways of working with git, as a team of developers

- https://www.atlassian.com/git/tutorials/comparing-workflows

BEWARE!

Beware! Git can provide some powerful commands. For an example of one potential disaster (using push --force), read this:

- https://jenkins.io/blog/2013/11/25/summary-report-git-repository-disruption-incident-of-nov-10th/

EXERCISES

1. Fork a project from GitLab, and cloning it locally.
   - Then try rolling back to an earlier version
2. Check in an existing Java project created with your IDE. Set up an appropriate ignore list
3. Try git rm which removes files from a repository. Take note of how it works.
4. Load GitKraken, connect to an existing repository
   - Experiment with using the git commands form the GUI.
5. In pairs: Try having more than one person work on a repository:
   a. First person can create a test repository (one you don't mind losing!)
   b. Give your partner access to the repository.
   c. Start editing files together. Try and see if you can cause some conflicts and problems with the fields in the repository. Use this as an opportunity to figure out what happens when things go wrong, and why the workflows can help here.