

## Lab 05: Git

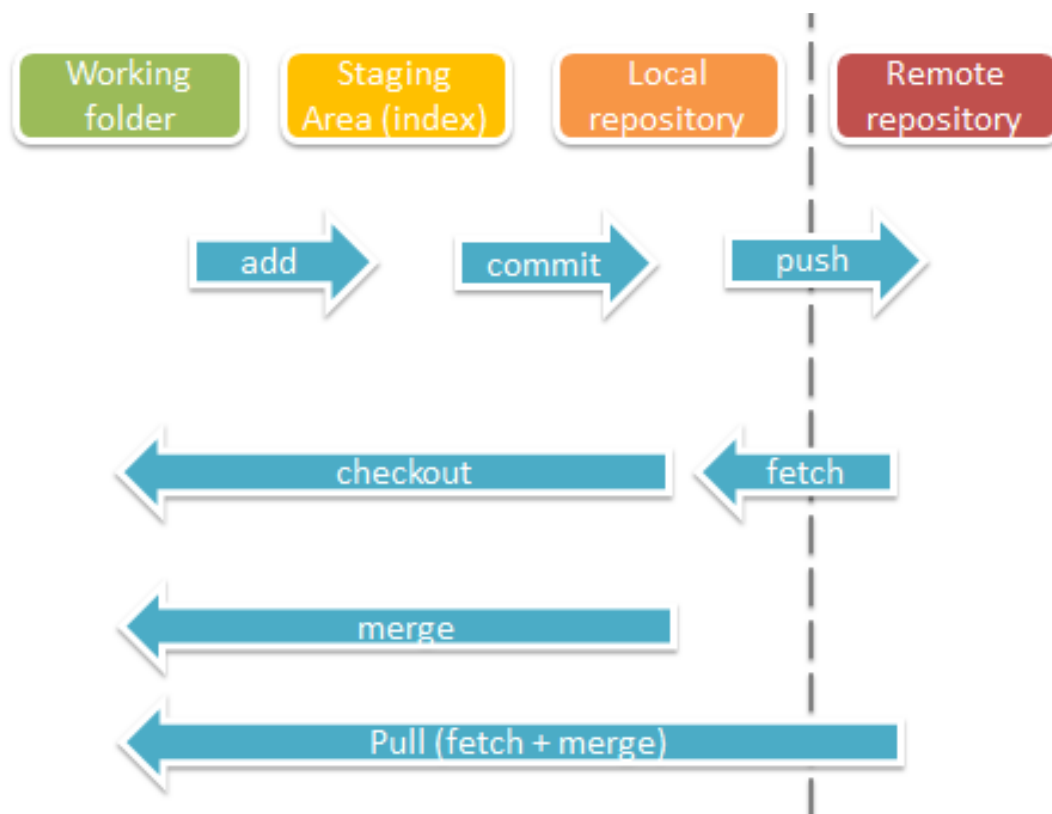
### Aims:

1. Set up your own Git repository project
  - a. Acquire more experience of using Git and source code versioning
  - b. Work with Gitlab
  - c. Set up SSH keys
2. Understand the differences between the local and remote repository and the relevant commands for each
3. Understand branches, and how to merge changes between them
4. Connect to existing repositories

\*Note: in this lab, we will use the school-provided git, named CSProject (<https://csprojects.nottingham.edu.cn>). For anyone who intends to work on the public git, e.g. Gitlab ([www.gitlab.com](http://www.gitlab.com)), this manual is equally effective. But please note that the coursework is based on CSProject.

---

This diagram may be useful to refer to:



## 1. Setting up local and remote repositories

In this lab, we will set up a local codebase, and a remote repository on CSProject.

On the standard desktop installation in the lab, you can use Git Bash to access Git from the command line. On your own Windows or macOS machine, you can go to <https://git-scm.com/downloads> to download the git installation packages, which, after installation, provides the Git Bash command line to you. (Note: During the installation, you can keep all default options and click 'Next', till completed.)

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 such as GIT GUI can be found from here <https://git-scm.com/downloads/guis>

Next you need to create a project space on the CSProject server. Go to <https://csprojects.nottingham.edu.cn/>. Log in, and create a New project, then choose "Create blank project": (1) Provide the project name and a URL name (z2019078 in my case); (2) Choose private or public depending on how you want it to be shared (To smoothly go through this lab, you can choose the public option. Open source project could be public; project for an enterprise could be private, etc.); (3) Put a readme into this project, so that it doesn't start empty.



## Create blank project

Create a blank project to store your files, plan your work, and collaborate on code, among other things.

### Project name

MyDMSLab24

Must start with a lowercase or uppercase letter, digit, emoji, or underscore. Can also contain dots, pluses, dashes, or spaces.

### Project URL

https://csprojects.n...




z2019078



### Project slug

mydmslab24

### Visibility Level

- ☐  Private  
Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.
- ☐  Internal  
The project can be accessed by any logged in user except external users.
- ☒  Public  
The project can be accessed without any authentication.

### Project Configuration


- ☒ Initialize repository with a README  
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.
- ☐ Enable Static Application Security Testing (SAST)  
Analyze your source code for known security vulnerabilities. [Learn more.](#)



Create project


Cancel

In the next page shown, observe that you now have one branch, which is the **master** trunk. Now click on the **readme.md** file, you may find that it contains the name of your project, and many getting started instructions – these are helpful information for you to refer to.


M

MyDMSLab24 


 

 Star 


0


 Fork 


0


More actions 


Project information


 1 Commit

 1 Branch

 0 Tags

 3 KiB Project Storage

 README

 Auto DevOps enabled

[+ Add LICENSE](#)

[+ Add CHANGELOG](#)



[+ Add CONTRIBUTING](#)


[+ Add Kubernetes cluster](#)

[+ Add Wiki](#)

[+ Configure Integrations](#)


Created on  
October 26, 2024


 main 


mydmslab24 / 


History



Find file

Edit 

Code 

 Initial commit  
Heng Yu authored in 4 hours

71b89b2d 

Name	Last commit	Last update
 README.md	Initial commit	in 4 hours
 README.md		

## MyDMSLab24

---

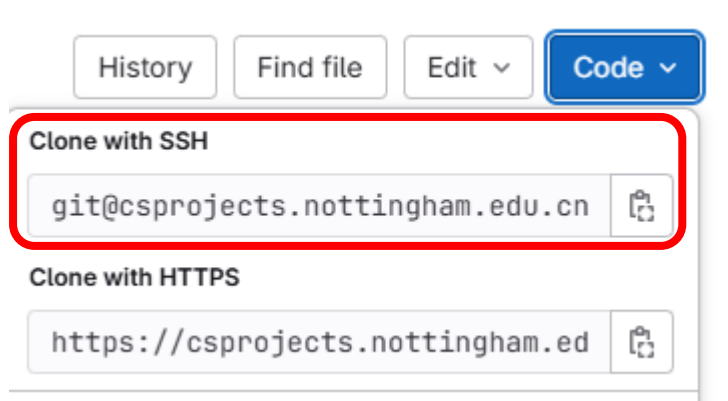
### Getting started

---

To clone the repo to your local machine, click the blue [Code](#) button, then copy the URL by clicking clone with SSH.

### IMPORTANT:

If you haven't configured an SSH to be used with GitLab, go to [Section 8](#) immediately! After setting up the SSH Keys, return here and continue.



Now open the Git Bash command line from your machine, create a local folder called 'git\_test' using `<mkdir git_test>`, then `cd` to 'git\_test' directory and enter:

`git clone <the address you just copied>`

and hit Enter. You will observe the following similar snapshot.

```
z2019078@UNNCITD2012-027 MINGW64 ~/git_test
$ git clone git@csprojects.nottingham.edu.cn:z2019078/mydmslab24.git
Cloning into 'mydmslab24'...
Enter passphrase for key '/c/Users/z2019078/.ssh/id_rsa':
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.

z2019078@UNNCITD2012-027 MINGW64 ~/git_test
$ ls
dmscsgit/  dmslab07/  mydmslab/  mydmslab24/
```

This shows that you have successfully cloned the remote repo into your local folder, which has the name "mydmslab24".

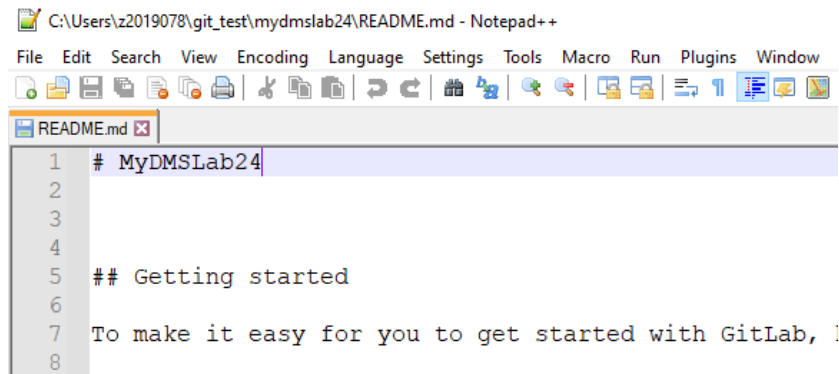
Enter this folder by typing: `cd mydmslab24/`, then `ls`

```
z2019078@UNNCITD2012-027 MINGW64 ~/git_test
$ cd mydmslab24/

z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (main)
$ ls
README.md
```

You find something interesting: (1) a 'main' label shows that this is the master branch you are currently dealing with; (2) the README.md file which you just clone to local. Use your favorite editor to open it, you see the content shown below, which is cloned from the remote repo.

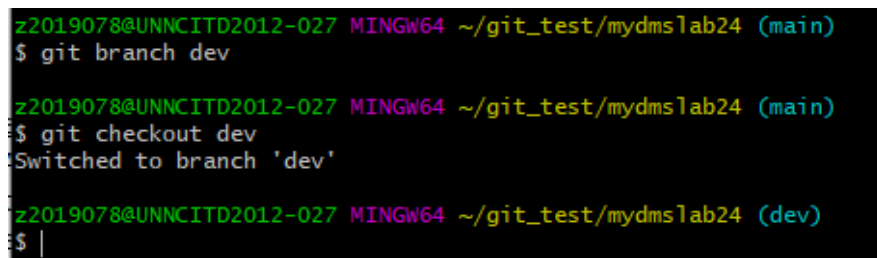
**Be careful! DO NOT EDIT THIS FILE NOW.**



```
C:\Users\z2019078\git_test\mydmslab24\README.md - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window
README.md
1 # MyDMSLab24
2
3
4
5 ## Getting started
6
7 To make it easy for you to get started with GitLab, !
8
```

## 2. Work on a branch

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:



```
z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (main)
$ git branch dev

z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (main)
$ git checkout dev
Switched to branch 'dev'

z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (dev)
$
```

Note the actions of branch and checkout:

```
git branch BRANCH_NAME      # create a new branch
git checkout BRANCH_NAME    # then switch to the new branch
```

===

**Tip:** use `git branch -d BRANCH_NAME` to delete the branch you created. Normally, a developer opens a new branch to work on new features. After completing the development and pushed+merged the branch on the remote server, the branch should normally be deleted.

===

Now you work in the dev branch. Open the README.md, and add a line “This is something I added to the branch ‘dev’”.

```
README.md x
1 # MyDMSLab24
2
3 This is something I added to the branch 'dev'
4
5 ## Getting started
6
7 To make it easy for you to get started with GitLab,
```

Save and close the editor, and apply **git add -A .** and **git commit** as shown below.

```
z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (dev)
$ git add -A .

z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (dev)
$ git commit -m "updated README.md in dev"
[dev 172abd6] updated README.md in dev
Committer: Heng Yu <z2019078@nottingham.edu.cn>
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, 1 insertion(+), 1 deletion(-)

z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (dev)
$ |
```

Now the README.md in the dev branch has been changed and logged.

===

**Tip:** You can quickly read the content of the first x lines of README.md using the “**head -n x README.md**” command.

```
z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (dev)
$ head -n 5 README.md
# MyDMSLab24

This is something I added to the branch 'dev'

## Getting started

z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (dev)
$ |
```

===

Now change back to the master branch, using:

```
git checkout main
```

===

**Task:** In the master branch, check whether the **README.md** file has been changed. If not, why it hasn't been changed?

===

More about **git branch** options and concepts, refer to the following links.

<https://git-scm.com/docs/git-checkout>

<https://www.atlassian.com/git/tutorials/using-branches/git-checkout#:~:text=Remember%20that%20the%20HEAD%20is,a%20%E2%80%99Cdetached%20HEAD%E2%80%99D%20state.>

### 3. Merging

When we are happy with our committed changes in `dev`, we want to merge them back into the `master` code set. There are a few options for this. One typical way is:

```
z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (dev)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (main)
$ git pull
Enter passphrase for key '/c/Users/z2019078/.ssh/id_rsa':
Already up to date.

z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (main)
$ git merge dev
Updating 71b89b2..172abd6
Fast-forward
 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (main)
$ head -n 5 README.md
# MyDMSLab24

This is something I added to the branch 'dev'

## Getting started

z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (main)
$ |
```

That is: switch to the master branch (`main`), pull any changes made to this branch from the server (i.e., to synchronize with the remote server first, in case someone else collaborating with you updates code in the server's master branch), then merge your `dev` branch into the master. You can check the content of **README.md** that it now intakes the contents from `dev`.

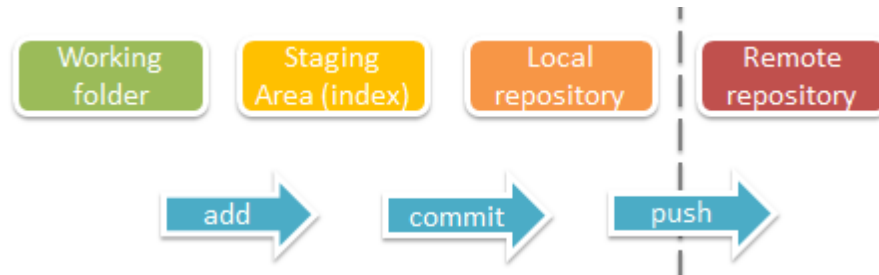


===

**Task:** Open the CSProject web UI to view your remote repo, refresh it and check the content of the README.md file again. Is the file's content changed?

===

Seems not yet! Because we have only committed to the **local repository**. Remember this:



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

```
z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (main)
$ git push
Enter passphrase for key '/c/Users/z2019078/.ssh/id_rsa':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 351 bytes | 351.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
To csprojects.nottingham.edu.cn:z2019078/mydmslab24.git
 71b89b2..172abd6  main -> main

z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (main)
$ |
```

Now refresh the GitLab page and check the **README.md** file. What do you see?

Name	Last commit	Last update
<a href="#">README.md</a>	updated README.md in dev	5 minutes ago

README.md

## MyDMSLab24

This is something I added to the branch 'dev'

### Getting started

Your commit message when merging.

- 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 Section 5, the link of workflow for more thorough examples of using git.

## 4. Ignoring files

Up to now we've talked about checking in source files or text files. There is 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-in 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> for more details)

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):

```
z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (main)
$ touch .gitignore

z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (main)
$ echo "*.class">.gitignore

z2019078@UNNCITD2012-027 MINGW64 ~/git_test/mydmslab24 (main)
$ cat .gitignore
*.class
```

## 5. 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/centralized-workflow>

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.

## 6. 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

<https://try.github.io/levels/1/challenges/1>

## 7. BEWARE!

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

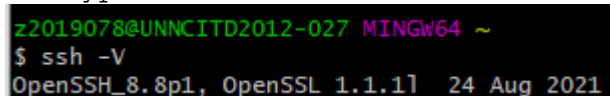
<http://jenkins-ci.org/content/summary-report-git-repository-disruption-incident-nov-10th>

## 8. SSH

In order for Git to know who it is talking to, you need to setup an SSH key locally, and add this to the Git system for the remote repository to work.

### Preparation

First of all, check whether your system has installed the OpenSSH with the correct version. Normally, OpenSSH comes pre-installed with macOS, Linux, and Windows 10. Open a system prompt (Windows users can use Git Bash), and type `ssh -V`



```
z2019078@UNNCITD2012-027 MINGW64 ~
$ ssh -V
OpenSSH_8.8p1, OpenSSL 1.1.1f 24 Aug 2021
```

Note that my version could be older than yours if you newly installed Git. It is preferred that the version be higher than 7.8. If your ssh version is below or equal to 7.8, refer to this page (<https://docs.gitlab.com/ce/ssh/README.html#rsa-keys-and-openssh-from-versions-65-to-78>)

### Generating a new RSA SSH Key

(If you already have an SSH key, find them usually in user's home directory, in the .ssh/ directory. Then this subsection can be skipped.)

Type in the following:

```
z2019078@UNNCITD2012-027 MINGW64 ~  
$ ssh-keygen -t rsa -b 2048 -C "heng.yu@nottingham.edu.cn"
```

**\*Note:** you are suggested to replace my email address with your own email address.

Press Enter, the following prompt will show.

```
z2019078@UNNCITD2012-027 MINGW64 ~  
$ ssh-keygen -t rsa -b 2048 -C "heng.yu@nottingham.edu.cn"  
Generating public/private rsa key pair.  
Enter file in which to save the key (/c/Users/z2019078/.ssh/id_rsa):
```

If you don't have a previously existed .ssh/ directory in your user home directory (i.e., this is the first time you creating the SSH), then just press Enter. Otherwise, you can choose to save it in another directory.

```
z2019078@UNNCITD2012-027 MINGW64 ~  
$ ssh-keygen -t rsa -b 2048 -C "heng.yu@nottingham.edu.cn"  
Generating public/private rsa key pair.  
Enter file in which to save the key (/c/Users/z2019078/.ssh/id_rsa):  
/c/Users/z2019078/.ssh/id_rsa already exists.  
Overwrite (y/n)? y  
Enter passphrase (empty for no passphrase):
```

Then, after entering the passphrase twice, your SSH keys (id\_rsa and id\_rsa.pub) should be generated.

### Add SSH to GitLab account

Copy your key to the clip board.

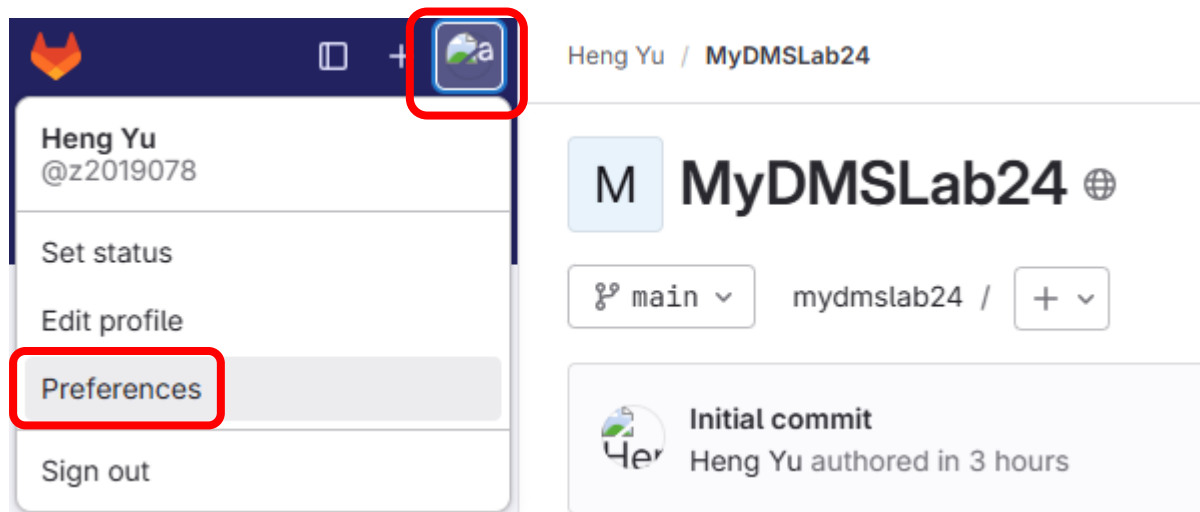
For Windows, type `cat ~/.ssh/id_rsa.pub | clip`

```
z2019078@UNNCITD2012-027 MINGW64 ~  
$ cat ~/.ssh/id_rsa.pub | clip
```

For Linux, type `xclip -sel clip < ~/.ssh/id_rsa.pub`

For macOS, type `pbcopy < ~/.ssh/id_rsa.pub`

Go to your Git, click your avatar on the upper right corner, then choose Preference.



On the left bar, find an item called SSH Keys, click it. Click 'Add new key' and paste the id\_rsa.pub into the Key box.



Make sure the title is automatically filled with your unique label, which is your email address. (This is the `-C` option in `ssh-keygen`). Then, press the “Add key”.

## Add an SSH key

Add an SSH key for secure access to GitLab. [Learn more.](#)

### Key

ssh-rsa

Hidden.

Begins with 'ssh-rsa', 'ssh-dss', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521@openssh.com'.

### Title

heng.yu@nottingham.edu.cn

Key titles are publicly visible.

### Usage type

Authentication & Signing

### Expiration date

2025-05-01



Optional but recommended. If set, key becomes invalid on the specified date.

Add key

Cancel

## Verify SSH Key working with GitLab

Run the following command in the terminal, to connect to csprojects.

```
z2019078@UNNCITD2012-027 MINGW64 ~  
$ ssh -T git@csprojects.nottingham.edu.cn
```

Enter your password, then if you see the following, you are all set!

```
z2019078@UNNCITD2012-027 MINGW64 ~  
$ ssh -T git@csprojects.nottingham.edu.cn  
Enter passphrase for key '/c/Users/z2019078/.ssh/id_rsa':  
Welcome to GitLab, @z2019078!
```

For more details about the SSH Key setup, refer to the following pages.

<http://doc.gitlab.com/ce/ssh/README.html>

<https://projects.cs.nott.ac.uk/help/ssh/README.html> (UNNC domain required)

You should keep your SSH key private! It allows you to communicate with your repository, so if someone else has it, your account is theirs.

## 9. EXERCISES

1. Try forking a project from GitHub, and cloning it locally.
  - a. Then try rolling back to an earlier version
2. Try checking in an existing Java project created with IntelliJ. Set up an appropriate ignore list.  
<https://www.jetbrains.com/help/idea/using-git-integration.html>  
<https://courses.cs.washington.edu/courses/cse373/19wi/resources/intellij/git/>  
<https://www.logicbig.com/tutorials/misc/git/intellij.html>
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 from 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 files in the repository. Use this as an opportunity to figure out what happens when things go wrong, and why the workflows can help here.