

# *Object Oriented Programming* **Environment Setup**

Shuo-Han Chen (陳碩漢),  
[shchen@ntut.edu.tw](mailto:shchen@ntut.edu.tw)

The Sixth Teaching Building 327  
M 15:10 - 16:00 & F 10:10 - 12:00

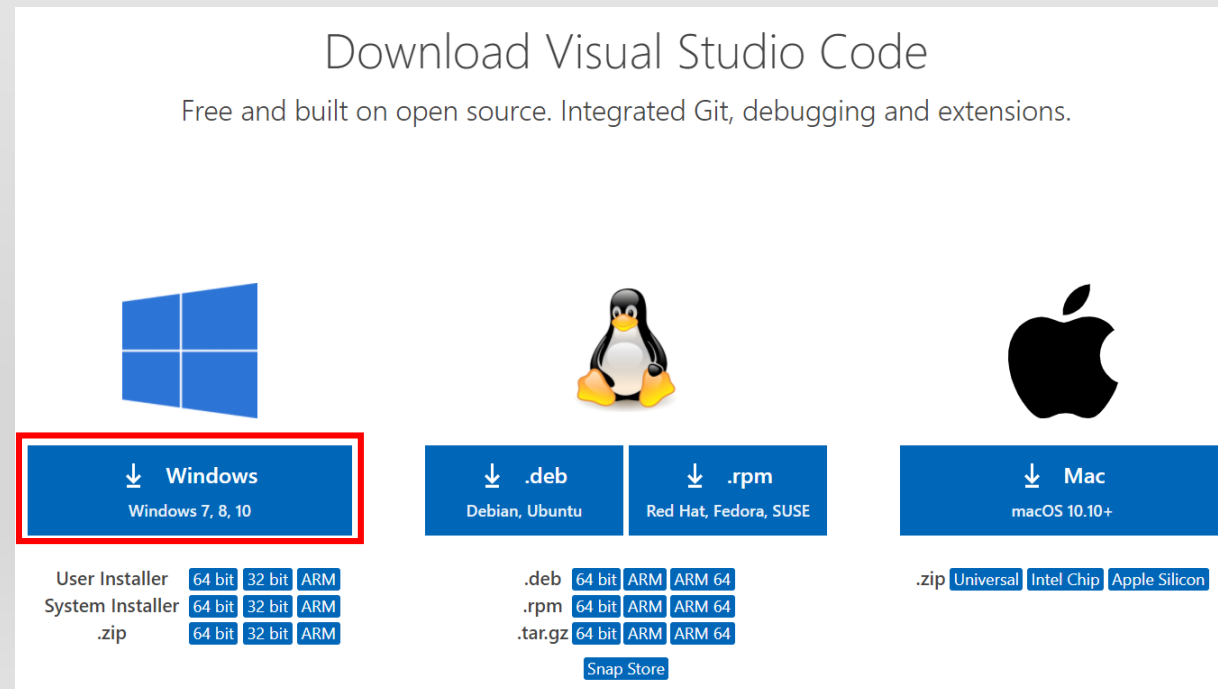
# Steps of Environment Setup

- Following steps are required to do your future homework
  - Part I -> Submit report of the success screenshot
    1. Install Visual Studio Code
    2. Setup ubuntu WSL & Google Test Library on Windows
    3. Do the HelloWorld
    4. Try using Google Test for your function
  - Part II -> Trigger Jenkins
    1. Go check GitLab and Jenkins websites
    2. Setup ssh key for git and Using git cmd
    3. Git push the HelloWorld

# Part I – Before 09/29 24:00

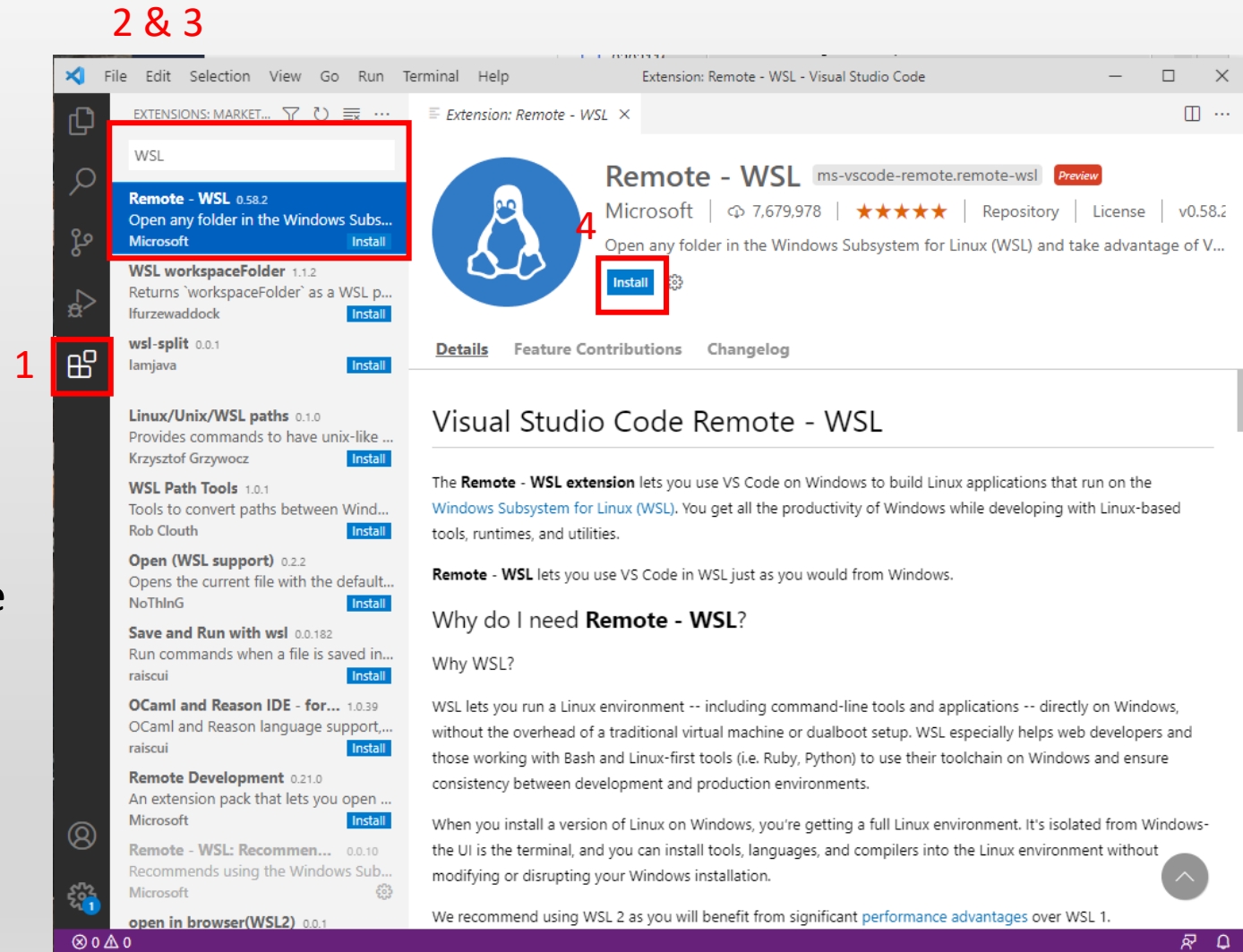
# Install Visual Studio Code

- You will find that every teacher ask you to use different Integrated Development Environment (IDE)
- It's quire normal since every company use different ways for writing code
- And, setup the environment is always the first thing for programmer
- Go here and download <https://code.visualstudio.com/download>



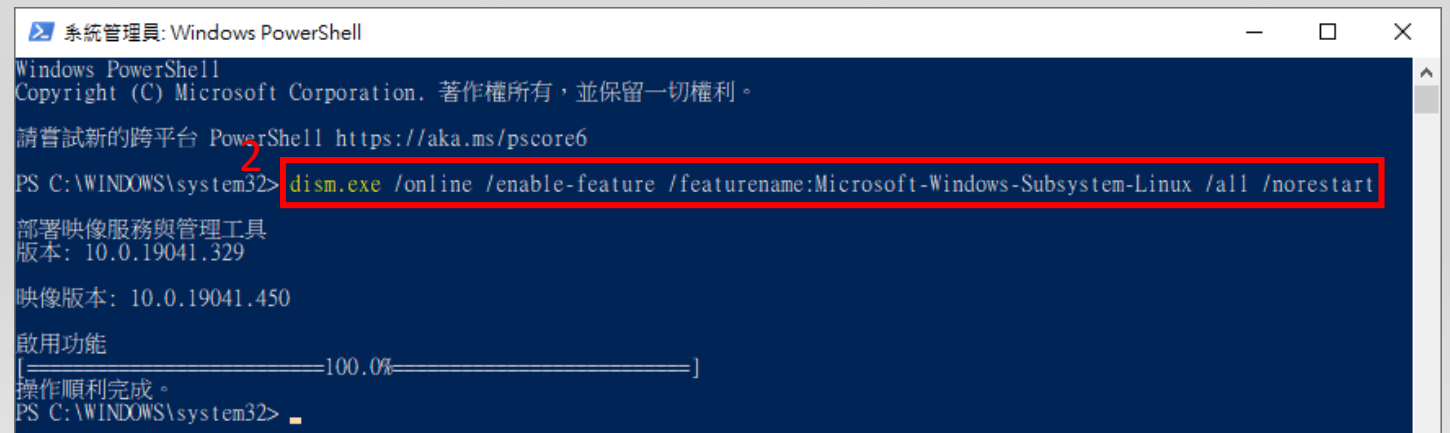
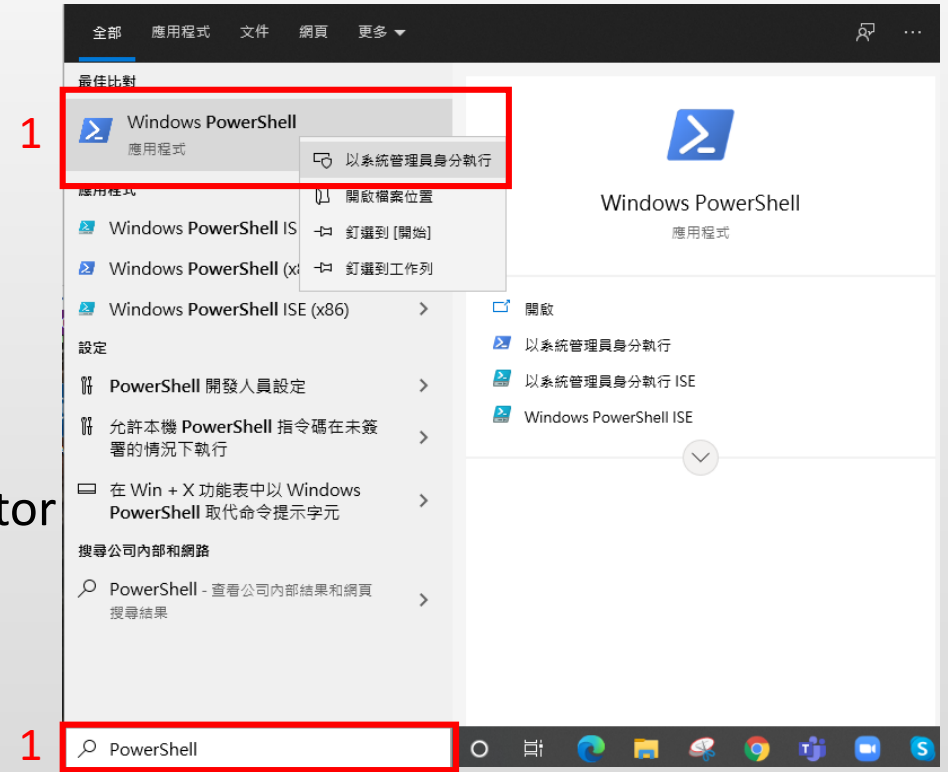
# Install Visual Studio Code (Cont'd)

- Install an extension in VS Code
  1. Select Extension
  2. Search “WSL”
  3. Find “Remote - WSL”
  4. Click Install
- What is WSL ?
  - WSL = Windows Subsystem for Linux
  - A virtual machine in Windows that have the functionality of Linux
- Why do we need this ?
  - We are going to compile your program with Linux commands
  - Key terms you can look into :
    - gcc, g++, make, makefile



# Setup Ubuntu Bash Shell on Windows

- We need ubuntu for
  - Compiler: g++
  - Builder: make and makefile
- We need to enable the “Windows Subsystem for Linux 1” on Windows 10
  1. Search & Right Click on PowerShell -> Run as Administrator
  2. Entry command: `dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart`
- Blank space could be missing
- Compare the your copied command with the one in the figure



# Setup Ubuntu Bash Shell on Windows (Cont'd)

- Go to Microsoft Store and Install Ubuntu 18.04 LTS

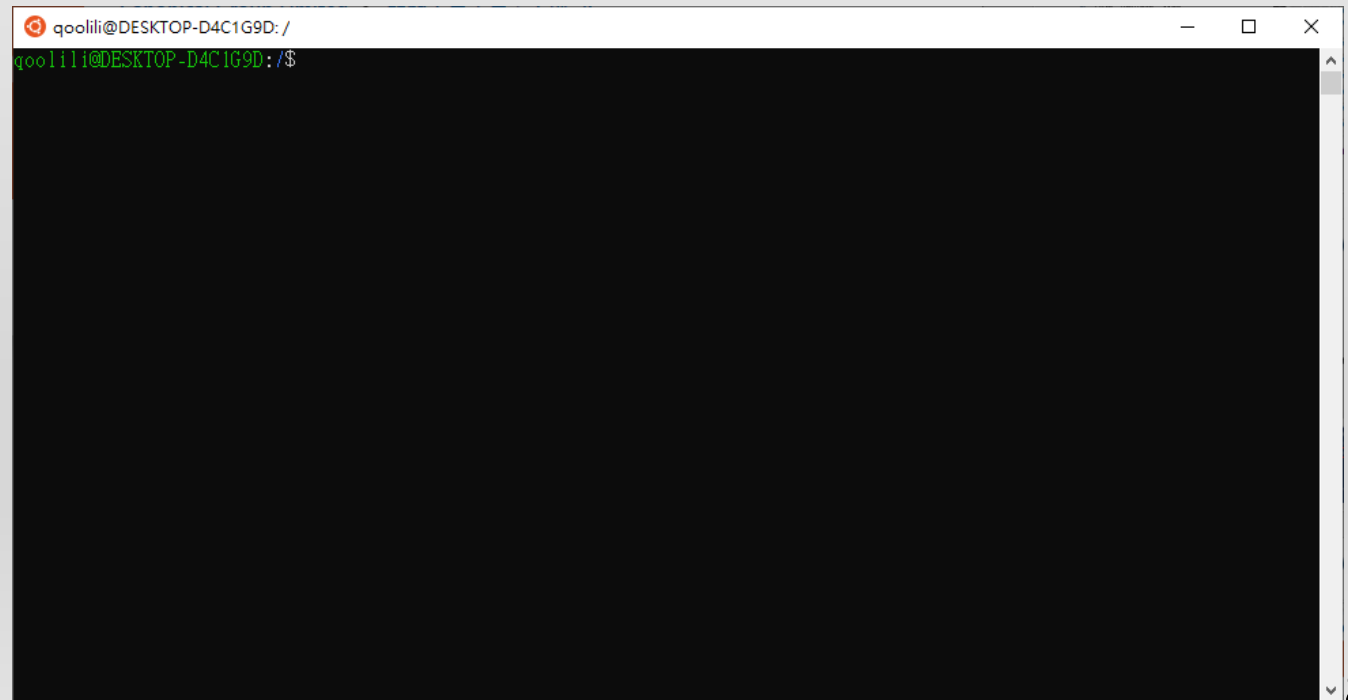
1. Search “ubuntu”
2. Select Ubuntu 18.04 LTS
3. Click “Install”
4. Wait until it's installed
5. Restart your computer
6. Run ubuntu & Complete setup

The collage consists of four screenshots illustrating the installation process:

- Top Left:** Microsoft Store search results for "ubuntu". A red box labeled "1" highlights the search bar containing "ubuntu". A red box labeled "2" highlights the "Ubuntu 18.04 LTS" application tile.
- Top Right:** Microsoft Store page for "Ubuntu 18.04 LTS". A red box labeled "3" highlights the "安裝" (Install) button. A red box labeled "5" highlights the "啟動" (Run) button.
- Bottom Left:** Windows File Explorer showing the "bash" application in the "快速存取" (QuickTime) section. A red box labeled "5" highlights the "bash" application.
- Bottom Right:** Windows File Explorer showing the "bash" application in the "快速存取" (QuickTime) section. A red box labeled "5" highlights the "bash" application.

# Setup Ubuntu Bash Shell on Windows (Cont'd)

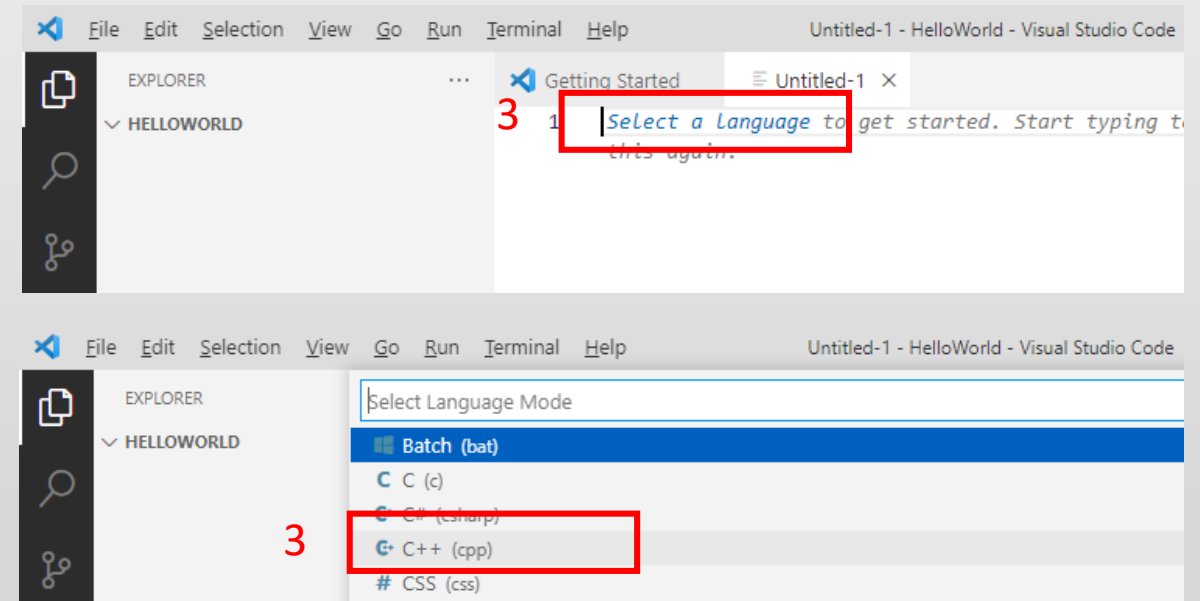
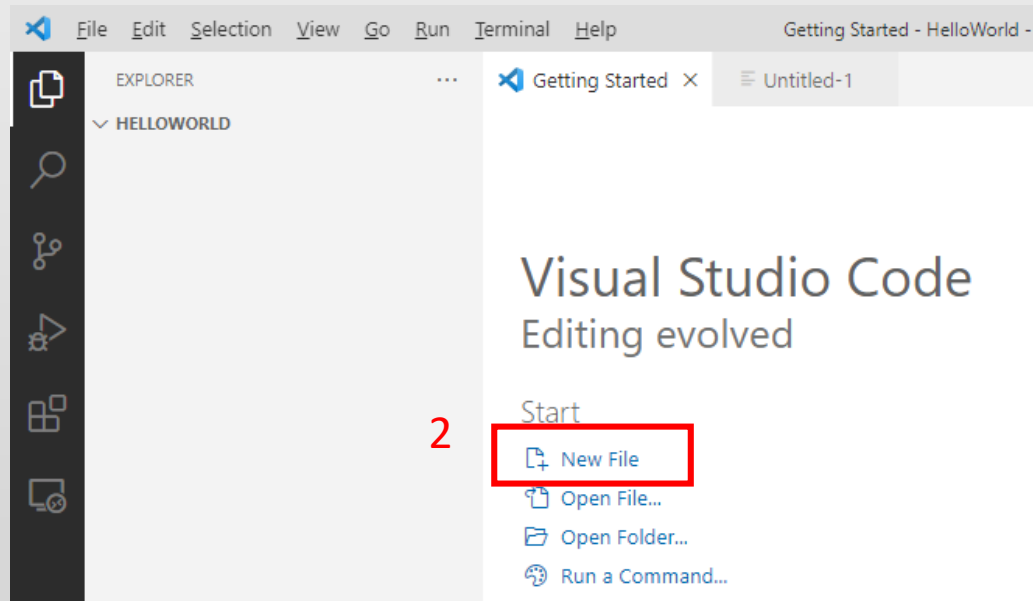
- Setting up Ubuntu 18.04 LTS may require to set username & password
- After Ubuntu 18.04 LTS is setup, you should see a black terminal
- Let's install the tools we need by entering following command
  1. `sudo apt-get update`
  2. `sudo apt-get install g++ make libgtest-dev cmake`
  3. `cd /usr/src/gtest`
  4. `sudo cmake CMakeLists.txt`
  5. `sudo make`
  6. `sudo cp *.a /usr/lib`
- libgtest-dev is the google test library





# Do the HelloWorld

1. Create a folder with the name of HelloWorld
2. Go back to VS code -> Open a folder -> Find the HelloWorld folder
3. Select New File -> Select Language "C++"



# Do the HelloWorld (Cont'd)

4. Start your HelloWorld coding
5. Save file by pressing **Ctrl** and **s** on your keyboard
6. Enter the file name as “HelloWorld.cpp”



```
g++ HelloWorld.cpp x
g++ HelloWorld.cpp > main()
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      cout << "Hello World! \n";
7      return 0;
8  }
```

- How do we run the program ? Where is the compile and run button ?
  - There is no such thing in large scale programming
  - Imagine you're now working at Google, there will be no compile and run button at all
  - Your code are integrated to the beta/release code through **continuous integration**

# Do the HelloWorld (Cont'd)

- Before we jump into continuous integration, let's try running the code locally

1. Create another new file -> Don't Select language
2. Copy and Paste the following
  - Add a TAB before g++ and rm. Three lines that need tabs.
3. Save the file with filename "makefile" **without extension**
4. Click Terminal -> New Terminal
  - You should see the terminal
  - If you didn't see this, click "+" & "power shell"

```
# This is the default target, which will be built when you invoke
make
.PHONY: all

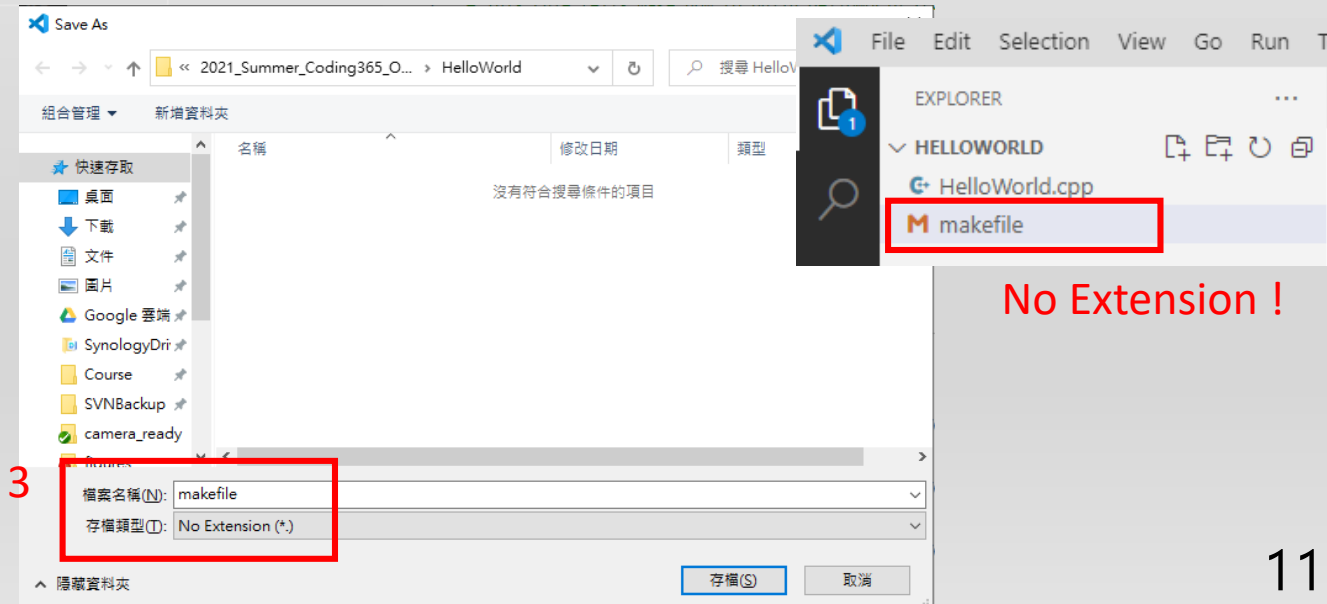
# This rule tells make how to build HelloWorld from
HelloWorld.cpp
all: HelloWorld.cpp
    g++ HelloWorld.cpp -o HelloWorld

# This rule tells make to delete hello and hello.o
.PHONY: clean
clean:
    rm -f HelloWorld
    rm -f ut_all
```



5. Enter "bash" to enter WSL

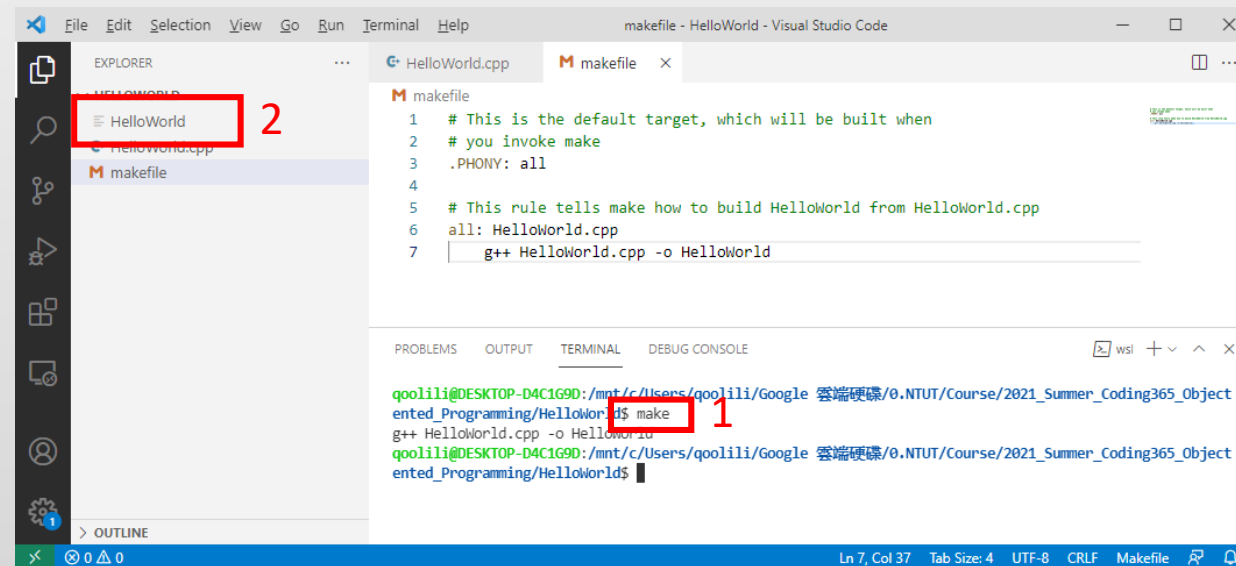
```
PS C:\Users\qoolili\Google 雲端硬碟\0.NTUT\Course\2021_Summer_Coding365_Object_Oriented_Programming\HelloWorld> bash
qoolili@DESKTOP-D4C1G9D:/mnt/c/Users/qoolili/Google 雲端硬碟/0.NTUT/2021_Summer_Coding365_Object_Oriented_Programming/HelloWorld$
```



No Extension !

# Do the HelloWorld (Cont'd)

- Let's compile and run your code locally -> At least you know your code can compile 😊
  - Type in cmd "make"
  - A runnable file will be generated



The screenshot shows the Visual Studio Code interface. In the Explorer panel on the left, the file 'HelloWorld' is highlighted with a red box and a red number '2'. The main editor shows the 'makefile' with the following content:

```
makefile
1 # This is the default target, which will be built when
2 # you invoke make
3 .PHONY: all
4
5 # This rule tells make how to build HelloWorld from HelloWorld.cpp
6 all: HelloWorld.cpp
7     g++ HelloWorld.cpp -o HelloWorld
```

The TERMINAL panel at the bottom shows the command 'make' being executed, highlighted with a red box and a red number '1'. The output of the command is:

```
qoolili@DESKTOP-D4C1G9D:/mnt/c/Users/qoolili/Google 雲端硬碟/0.NTUT/Course/2021_Summer_Coding365_Object
ented_Programming/HelloWorld$ make
g++ HelloWorld.cpp -o HelloWorld
qoolili@DESKTOP-D4C1G9D:/mnt/c/Users/qoolili/Google 雲端硬碟/0.NTUT/Course/2021_Summer_Coding365_Object
ented_Programming/HelloWorld$
```

- Run the compiled file by typing in "./HelloWorld"



The screenshot shows the terminal output after running the compiled program. The command './HelloWorld' is highlighted with a red box. The output is:

```
qoolili@DESKTOP-D4C1G9D:/mnt/c/Users/qoolili/Google 雲端硬碟/0.NTUT/Course/2021_Summer_Coding365_Object
ented_Programming/HelloWorld$ ./HelloWorld
Hello World!
```

- If you see the output, then Congrats ! We're half way there.

# Do the HelloWorld (Cont'd)

- Have problem with makefile?
  - It's usually the issue of missing a "tab" before gcc & rm
  - Do the following to double check
  - In cmd
    - `cat -e -t -v makefile`
  - Make sure there is a "^I" before gcc & rm
  - Try entering "tab" a few time, it might not show up at first time
- Ref:  
<https://stackoverflow.com/questions/16931770/makefile4-missing-separator-stop>



make has a very stupid relationship with tabs. All actions of every rule are identified by tabs. And, no, four spaces don't make a tab. Only a tab makes a tab.

To check, I use the command `cat -e -t -v makefile_name`.

It shows the presence of tabs with `^I` and line endings with `$`. Both are vital to ensure that dependencies end properly and tabs mark the action for the rules so that they are easily identifiable to the make utility.

Example:

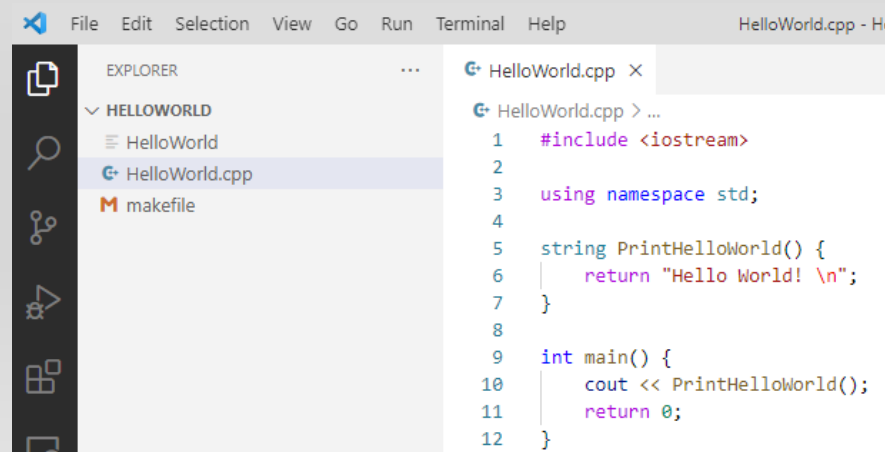
```
all:ll$      ## here the $ is end of line ...
$
ll:ll.c      $
^Igcc -c -Wall -Werror -O2 c.c ll.c -o ll $@ $<$
## the ^I above means a tab was there before the action part, so this line is ok
.
$
clean :$
    \rm -fr ll$
## see here there is no ^I which means , tab is not present ....
## in this case you need to open the file again and edit/ensure a tab
## starts the action part
```

# Try Google Test Your Code

- As we already discussed, big company usually intergrade your code to the beta/release code through **continuous integration**
- Before integration, **testing** need to be carried out
  - Make sure your code will do what it aims to
  - Make sure your code will not mess up existing code
  - This is very common, even for senior engineers ! So, always do testing !
- In this course, we will use google test library for testing your **functions**.

## A. Let's rewrite your HelloWorld to function-based from

1. Type “make” to recompile
2. Run the compiled file again  
-> “./HelloWorld”
3. Make sure the result is the same



The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane shows a project named 'HELLOWORLD' with files 'HelloWorld' and 'makefile'. The main editor displays 'HelloWorld.cpp' with the following code:

```
1 #include <iostream>
2
3 using namespace std;
4
5 string PrintHelloWorld() {
6     return "Hello World! \n";
7 }
8
9 int main() {
10     cout << PrintHelloWorld();
11     return 0;
12 }
```

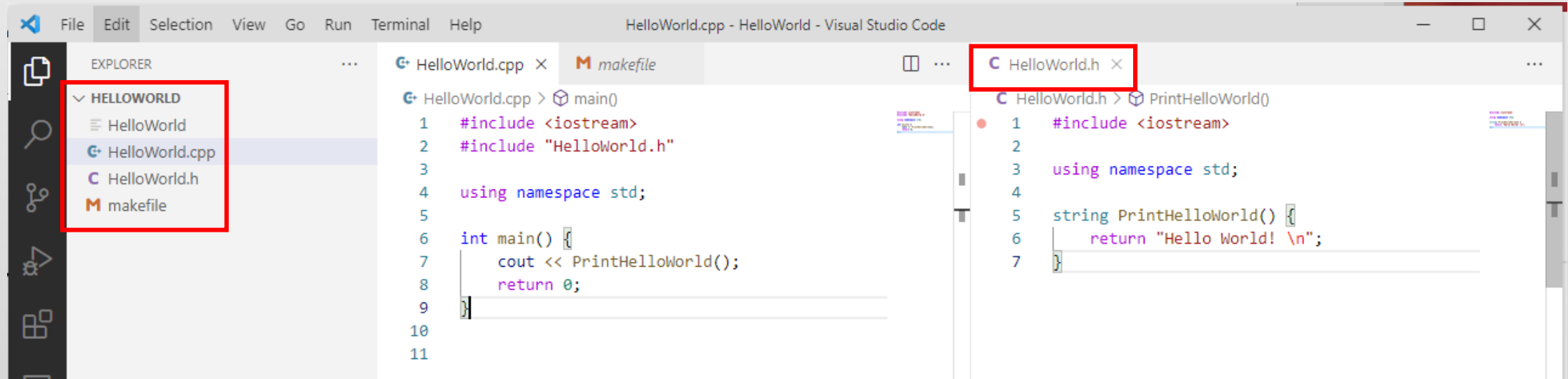


The screenshot shows a terminal window with the following commands and output:

```
qoolili@DESKTOP-D4C1G9D:/mnt/c/Users/qoolili/Google 雲端硬碟/0.NT
course/2021_Summer_Coding365_Object_Oriented_Programming/Helloworl
d$ make
g++ HelloWorld.cpp -o HelloWorld
qoolili@DESKTOP-D4C1G9D:/mnt/c/Users/qoolili/Google 雲端硬碟/0.NT
course/2021_Summer_Coding365_Object_Oriented_Programming/Helloworl
d$ ./HelloWorld
Hello World!
```

# Try Google Test Your Code (Cont'd)

- B. Move the function you just write to “HelloWorld.h” and include “HelloWorld.h” in “HelloWorld.c”
- Recompile & run again to make sure everything works fine



# Try Google Test Your Code (Cont'd)

## c. Prepare Google Test related code

1. New a file -> “ut\_main.cpp” and Prepare the content as follows
2. Update the makefile as follows
3. Try make and run by typing in “./ut\_all”

The screenshot displays the Visual Studio Code interface with three main components highlighted by red boxes and numbered 1, 2, and 3.

**Box 1: ut\_main.cpp**

```
1 #include <gtest/gtest.h>
2 #include "HelloWorld.h"
3
4 TEST(HELLOWORLD, PrintHelloWorld) {
5     string output = PrintHelloWorld();
6     string compare = "Hello World! \n";
7     ASSERT_EQ(output, compare);
8 }
9
10 int main(int argc, char **argv){
11     testing::InitGoogleTest(&argc, argv);
12     return RUN_ALL_TESTS();
13 }
14
```

**Box 2: Makefile**

```
1 # This is the default target, which will be built when
2 # you invoke make
3 .PHONY: all
4
5 # Redefine the target all with the requirement of hello & ut_all
6 all: hello ut_all
7
8 # This rule tells make how to build HelloWorld from HelloWorld.cpp
9 hello: HelloWorld.cpp
10     g++ HelloWorld.cpp -o HelloWorld
11
12 # This rule tells make how to build ut_all from ut_main.cpp
13 ut_all: ut_main.cpp
14     g++ -std=c++11 -Wfatal-errors ut_main.cpp -o ut_all -lgtest -lpthread
15
16 # This rule tells make to delete hello and hello.o
17 .PHONY: clean
18 clean:
19     rm -f HelloWorld
20     rm -f ut_all
21
```

**Box 3: Terminal Output**

```
qoolili@DESKTOP-D4C1G9D:/mnt/c/Users/qoolili/Google Drive/0.NTUT/Course/2021_Summer_Coding365_Object_Oriented_Programming/HelloWorld$ make
g++ HelloWorld.cpp -o HelloWorld
g++ -std=c++11 -Wfatal-errors ut_main.cpp -o ut_all -lgtest -lpthread
qoolili@DESKTOP-D4C1G9D:/mnt/c/Users/qoolili/Google Drive/0.NTUT/Course/2021_Summer_Coding365_Object_Oriented_Programming/HelloWorld$ ./ut_all
[=====] Running 1 test from 1 test case.
[-----] Global test environment set-up.
[-----] 1 test from HELLOWORLD
[ RUN     ] HELLOWORLD.PrintHelloWorld
[ OK      ] HELLOWORLD.PrintHelloWorld (0 ms)
[-----] 1 test from HELLOWORLD (1 ms total)

[-----] Global test environment tear-down
[=====] 1 test from 1 test case ran. (2 ms total)
[ PASSED ] 1 test.
qoolili@DESKTOP-D4C1G9D:/mnt/c/Users/qoolili/Google Drive/0.NTUT/Course/2021_Summer_Coding365_Object_Oriented_Programming/HelloWorld$
```



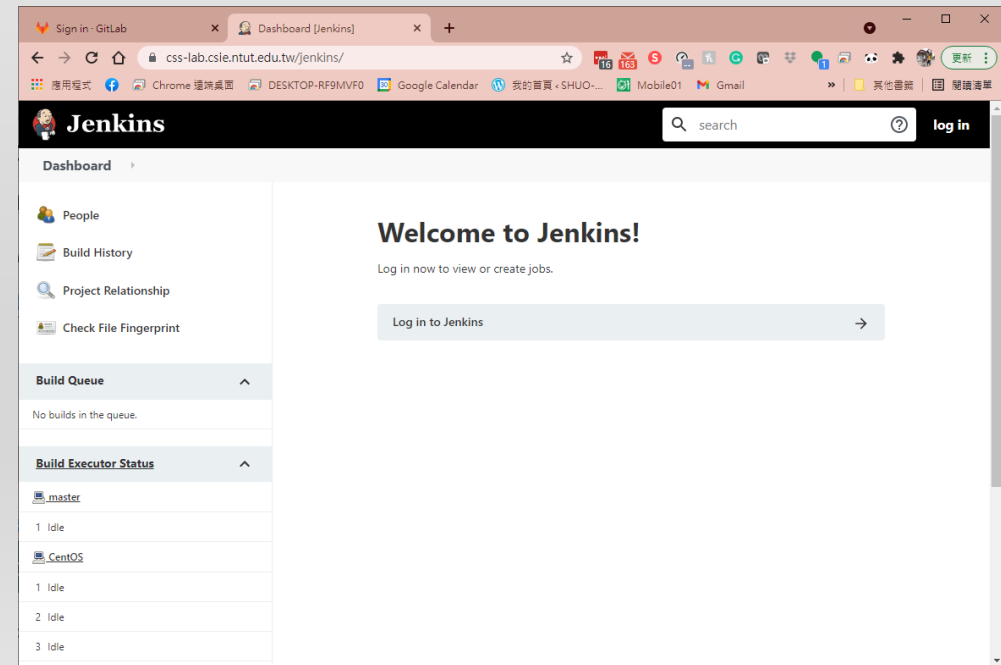
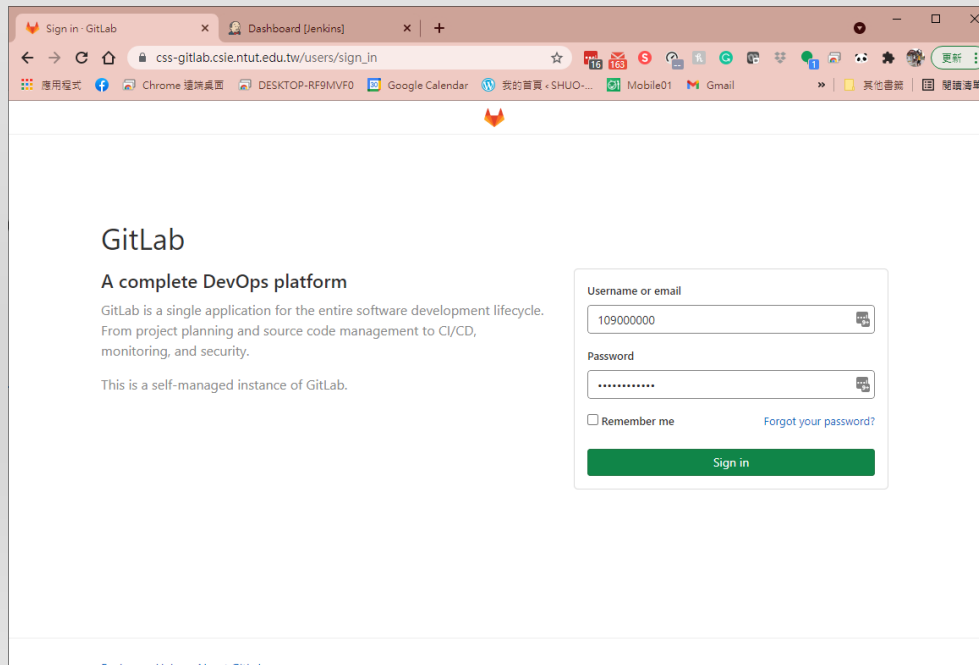
# Submit Report

- Please see the word file for report template
- **Strict format is enforced**
  - Content format: should be set with 16pt row height, align to the left, font size 12pt.
  - Caption format: 18pt and Bold font.
  - Font format: Times New Roman.
  - Figure: center with single line row height.
  - Change the title to your student ID and name in Chinese. If you don't has ID, just leave it blank.
  - Upload pdf file with the file name format : **OOP\_HW00\_1090000000.pdf** (change to your student ID) If you don't has ID, fill your name instead.
  - Remove the line starting with //.
  - Remove format guide part before uploading.

## Part II – Before 10/3 24:00

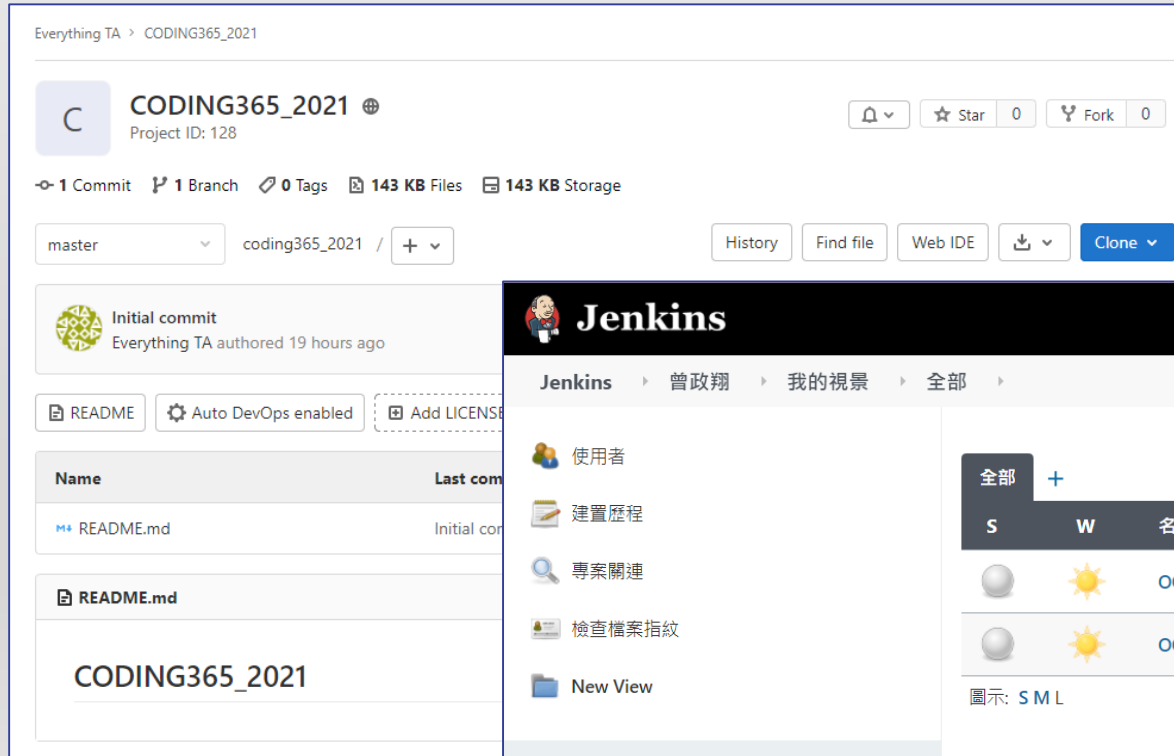
# Go check GitLab and Jenkins websites

- After testing our code locally, we now need to upload our code to GitLab server for triggering continuous integration (default password: 12345678. Change it after login, loss points if you did not change it TODAY.)
- Try login at following two websites
  - [https://css-gitlab.csie.ntut.edu.tw/users/sign\\_in](https://css-gitlab.csie.ntut.edu.tw/users/sign_in)
  - <https://css-lab.csie.ntut.edu.tw/jenkins/>




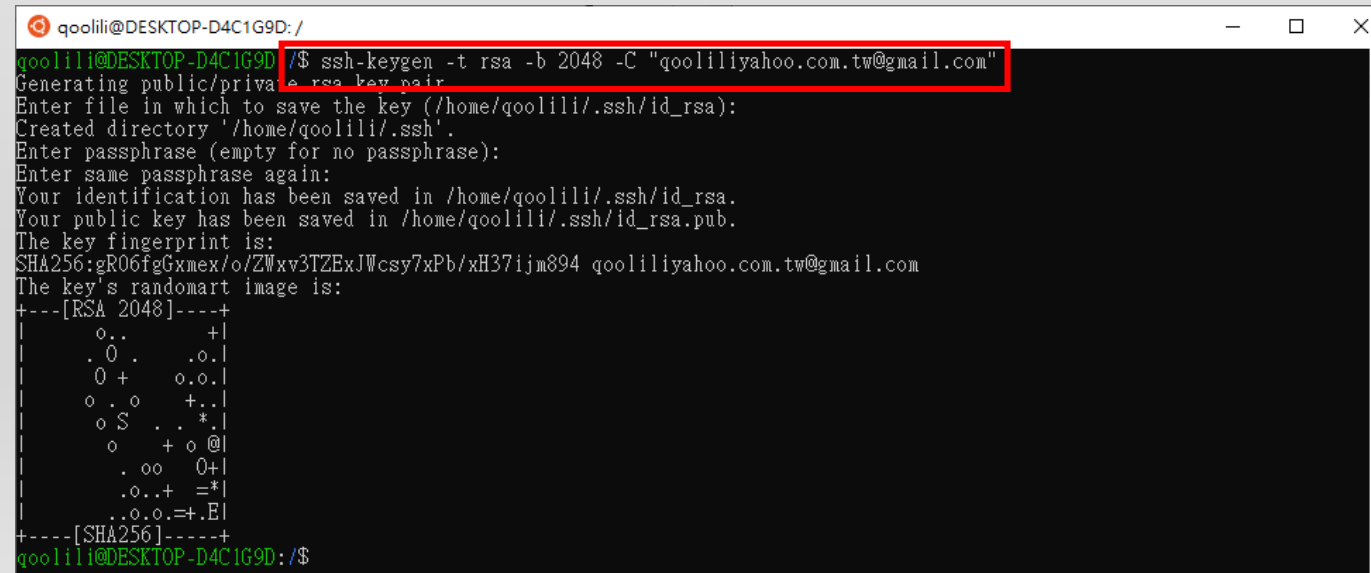
# Go check GitLab and Jenkins websites (Cont'd)

- All Setting has been set
  - On Gitlab, there is **one** repository. -> It's a remote repository for uploading your code
    - Git push to this repository will trigger jobs on Jenkins
  - On Jenkins, there is **two** jobs. -> Used for running testing on your code
    - HW will run the test cases you wrote. TA will run the test cases provided by us.



# Setup ssh key for git and Using git cmd

1. Generate a ssh key first locally in your bash terminal
  - If forget how to open this windows by searching, just search bash in 
2. Enter `ssh-keygen -t rsa -b 2048 -C "email@example.com"`
  - Change email@example.com to your email
  - Press enter for using default location
  - Leave the your passphrase (secret token) **BLANK**, then enter
  - Then, you should see the following



```
qoolili@DESKTOP-D4C1G9D: /
qoolili@DESKTOP-D4C1G9D: /$ ssh-keygen -t rsa -b 2048 -C "qooliliyahoo.com.tw@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/qoolili/.ssh/id_rsa):
Created directory '/home/qoolili/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/qoolili/.ssh/id_rsa.
Your public key has been saved in /home/qoolili/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:gR06fgGxmex/o/ZWxv3TZExJWcsy7xPb/xH37ijm894 qooliliyahoo.com.tw@gmail.com
The key's randomart image is:
+---[RSA 2048]---+
|  o..      +|
|  .O.      .o.|
| 0+   o.o.|
| o..o   +..|
| oS  . . *|
|  o   + o @|
|  . oo 0+|
|  .o..+ =*|
|  ..o.o.=+E|
+---[SHA256]-----+
qoolili@DESKTOP-D4C1G9D: /$
```

# Setup ssh key for git and Using git cmd (Cont'd)

3. Print the public key through `cat` command
  - The path of your public key can be found as follows

```
qoolili@DESKTOP-D4C1G9D: /
qoolili@DESKTOP-D4C1G9D:/$ ssh-keygen -t rsa -b 2048 -C "qooliliyahoo.com.tw@gmail.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/qoolili/.ssh/id_rsa):
Created directory '/home/qoolili/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/qoolili/.ssh/id_rsa.
Your public key has been saved in /home/qoolili/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:gr06feGxmex/o/ZWxv3TZEJWcsy7xPb/xH37ijm894 qooliliyahoo.com.tw@gmail.com
```

- Then, type `cat /home/{see your path above}/.ssh/id_rsa.pub`

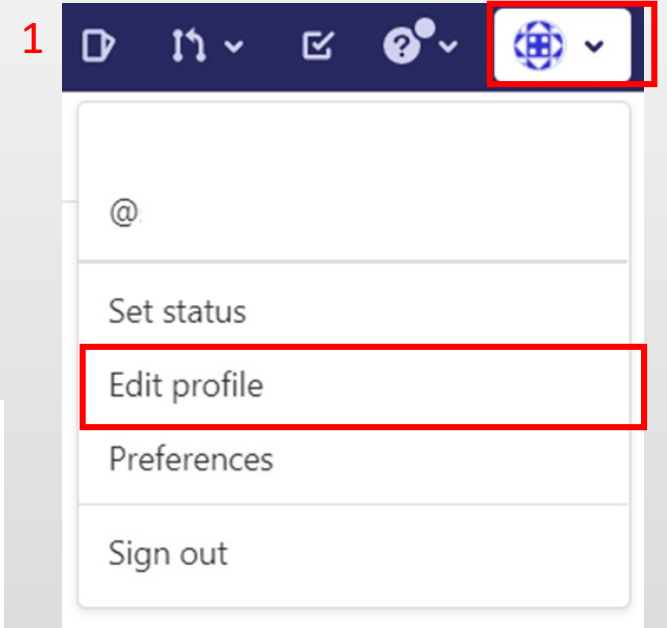
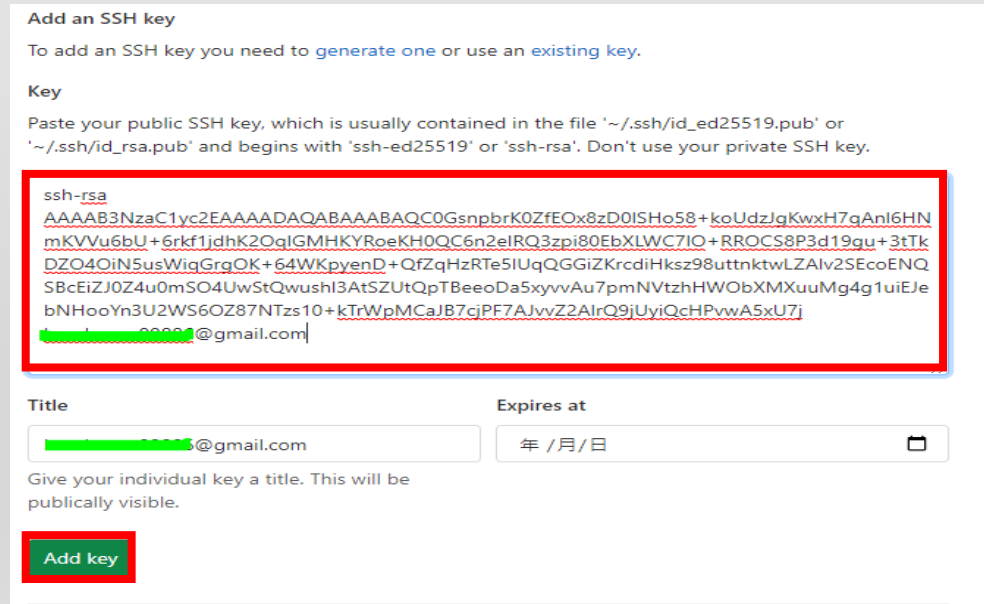
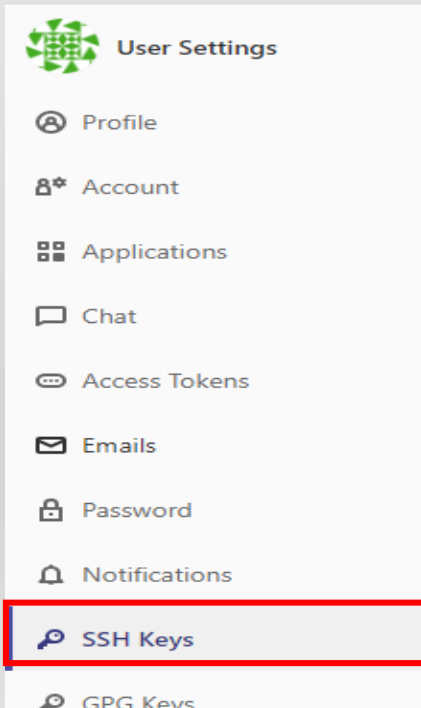
```
qoolili@DESKTOP-D4C1G9D: /
qoolili@DESKTOP-D4C1G9D:/$ cat /home/qoolili/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDMh7+EBP+kPjZHARXmnHuPWb00Rls989TnkzM6ls7p+DvkZMosWoyPbqS3LcpSpxRrbyMipw/ZF3btXnsU
G0drBXZXZ0JJjxFOzktwhOBqSWq1GMQGGbEWDxNsLj9br39afnoTb5scEBiWzcd18uyypeB5ADL1M8zcosEpTom5Xsmihlw4o4Z9kFHU6K++Z1yrTkV/4cCF
2PPed2tIqs0cgT3V6CbPgcg3pdJUicP00uwnuVAgoxTSkgWau0cPg3UW0tdR6lzlgbuc9vsSHI29yt9qGBUWr/41Q0DzBoQGzI/iwsnythko5y0CYhEK6Ic
76GKFZUsw7j6KAxmzpk7 qooliliyahoo.com.tw@gmail.com
qoolili@DESKTOP-D4C1G9D:/$
```

- Copy the text starting with `ssh-rsa` and ending with your email (Be careful of extra blank at the end)

```
qoolili@DESKTOP-D4C1G9D: /
qoolili@DESKTOP-D4C1G9D:/$ cat /home/qoolili/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDMh7+EBP+kPjZHARXmnHuPWb00Rls989TnkzM6ls7p+DvkZMosWoyPbqS3LcpSpxRrbyMipw/ZF3btXnsU
G0drBXZXZ0JJjxFOzktwhOBqSWq1GMQGGbEWDxNsLj9br39afnoTb5scEBiWzcd18uyypeB5ADL1M8zcosEpTom5Xsmihlw4o4Z9kFHU6K++Z1yrTkV/4cCF
2PPed2tIqs0cgT3V6CbPgcg3pdJUicP00uwnuVAgoxTSkgWau0cPg3UW0tdR6lzlgbuc9vsSHI29yt9qGBUWr/41Q0DzBoQGzI/iwsnythko5y0CYhEK6Ic
76GKFZUsw7j6KAxmzpk7 qooliliyahoo.com.tw@gmail.com
qoolili@DESKTOP-D4C1G9D:/$
```

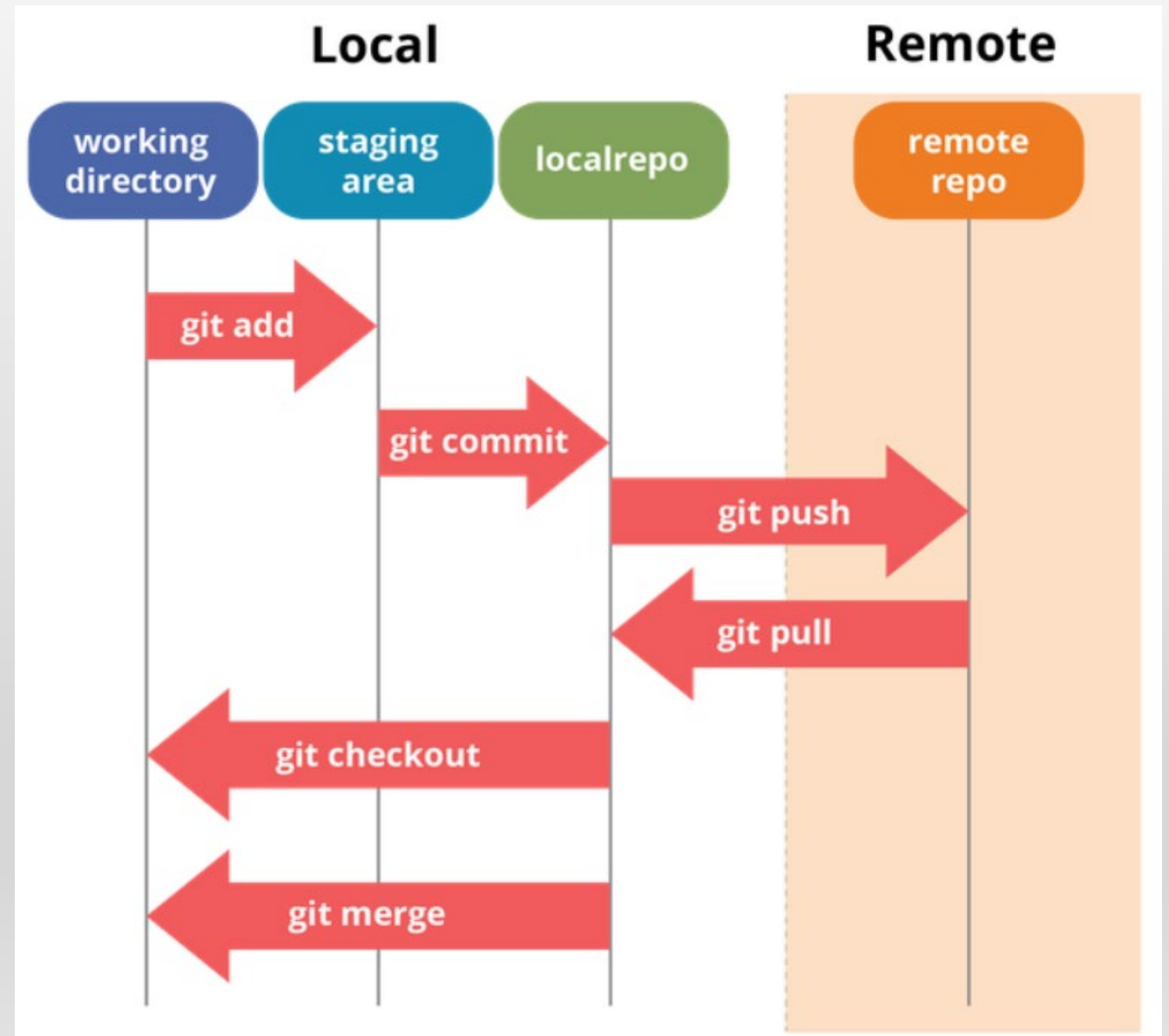
# Setup ssh key for git and Using git cmd (Cont'd)

1. On the right-upper corner, click your icon -> Settings/Edit Profile
2. Click “SSH Keys” on the left
3. Paste the text you copied to the text box, then click add key
4. You're all set



# Setup ssh key for git and Using git cmd (Cont'd)

- Using repository to track the revised history of files and folders
  - Local repository (本地)
  - Remote repository (遠端)





# Git push the HelloWorld

- If you haven't installed git, please follow ...

1. Open bash
2. Install git with -> `sudo apt-get install git`
3. Check you installation -> `git --version`

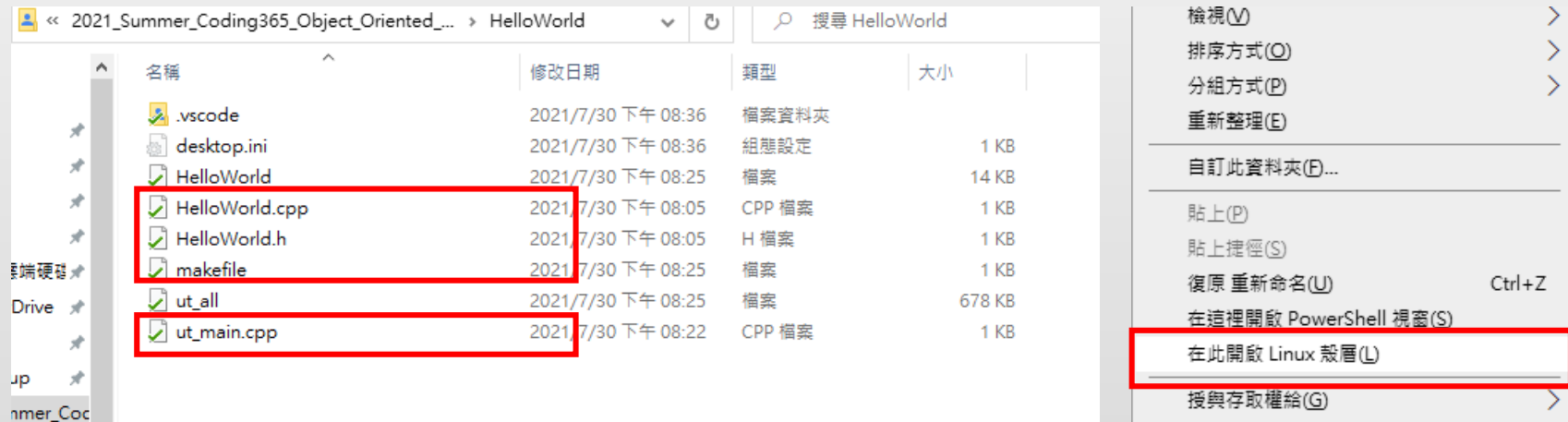
```
qoolili@DESKTOP-GOP9MSC:/mnt/c/Users/qoolili/Desktop/In-Class Projects$ git --version  
git version 2.25.1
```

4. Set information (Change the email and name)
  - > `git config --global user.email "t109000000@ntut.edu.tw"`
  - > `git config --global user.name "Shuo-Han Chen"`

```
qoolili@DESKTOP-GOP9MSC:/mnt/c/Users/qoolili/Desktop/In-Class Projects$ git config --global user.email "shchen@ntut.edu.tw"  
qoolili@DESKTOP-GOP9MSC:/mnt/c/Users/qoolili/Desktop/In-Class Projects$ git config --global user.name "Shuo-Han Chen"
```

# Git push the HelloWorld (Cont'd)

1. In your HelloWorld folder, press **shift** and **right click** at anywhere
2. Then, select Open Linux bash



2. Initialize local repository -> **sudo git init**

```
root@DESKTOP-0UM5M4J:/mnt/c/Users/qooli/Google 雲端硬碟/Course/2020_Fall_Object_Oriented_Programming/Pre-class Project/Xstring# git init
Initialized empty Git repository in /mnt/c/Users/qooli/Google 雲端硬碟/Course/2020_Fall_Object_Oriented_Programming/Pre-class Project/Xstring/.git/
```

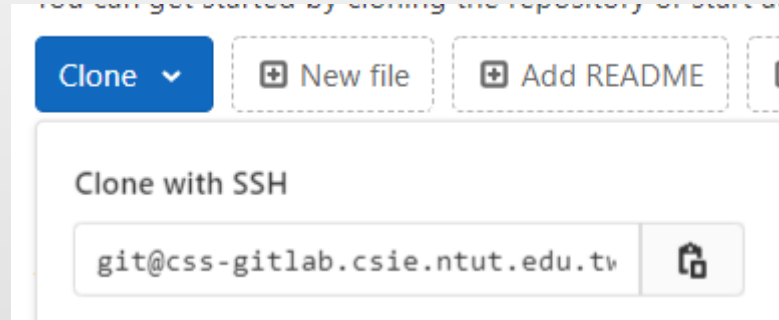
3. Add files to be uploaded into local repository -> **git add**

- Add only \*.cpp, \*.h, makefile

```
qoolili@DESKTOP-D4CIG9D:/mnt/c/Users/qoolili/Google 雲端硬碟/0.NTUT/Course/2021_Summer_Coding365_Object_Oriented_Programming/HelloWorld$ git add *.cpp *.h makefile
```

# Set Remote Repository

4. Copy your git link from our gitlab



5. Add remote -> `sudo git remote add origin git@....`

```
root@DESKTOP-0UM5M4J:/mnt/c/Users/qooli/Google 雲端硬碟/Course/Jenkins# git remote add origin https://css-gitlab.csie.ntut.edu.tw/qoolili/jenkinscript.git
```

# Git Commit & Push

6. Commit files to local repository -> `git commit -am "HW01"`
- -a : commit all changed files , -m “提交訊息” : specify commit message
  - This command will make changes to your local repository

```
root@DESKTOP-OUM5M4J:/mnt/c/Users/qooli/Google 雲端硬碟/Course/2020_Fall_Object_Oriented_Programming/Pre-class Project/Xstring# git commit -am "HW01"
[master (root-commit) fad1216] HW01
6 files changed, 124 insertions(+)
create mode 100644 makefile
create mode 100644 src/main.cpp
create mode 100644 src/xstring.cpp
create mode 100644 src/xstring.h
create mode 100644 test/ut_main.cpp
create mode 100644 test/ut_xstring.h
```

7. Push Files onto Gitlab Project -> `git push -u origin master`
- This command only used for the first-time push
  - Next time, you only need `git push`
  - Go to your project on our Gitlab, you should see files on your git
  - And it will automatically trigger Jenkins

# Git push the HelloWorld (Cont'd)

- Go check your result on Jenkins

專案 CODING365\_2021\_1090000000\_HW



工作區



最近變更



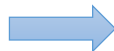
最新測試結果 (無失敗)

上游專案

OOP2020f\_1090000000\_HW

永久連結

- 最新建置 (#23), 25 分 以前
- 最新穩定建置 (#23), 25 分 以前
- 最新成功建置 (#23), 25 分 以前
- 最新失敗建置 (#19), 4 天 4 時 以前
- 最新不成功建置 (#19), 4 天 4 時 以前
- Last completed build (#23), 25 分 以前

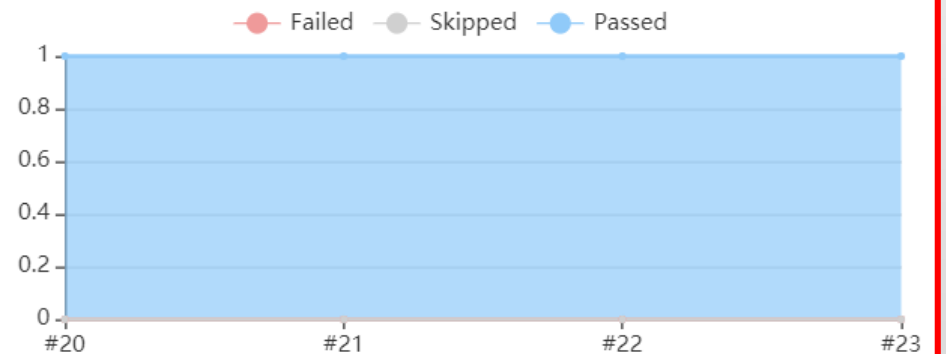


Console Output

新增描述

停用專案

測試結果趨勢



# Git push the HelloWorld (Cont'd)

- For now, only the job with HW is triggered. The job with TA will be used in the future.

## 終端機輸出

```
mkdir -p bin obj
g++ test/ut_main.cpp -o bin/ut_all -lgtest -lpthread
+ bin/ut_all --gtest_output=xml:result.xml
[=====] Running 1 test from 1 test suite.
[-----] Global test environment set-up.
[-----] 1 test from HelloWorld
[ RUN      ] HelloWorld.case1
[      OK  ] HelloWorld.case1 (0 ms)
[-----] 1 test from HelloWorld (0 ms total)

[-----] Global test environment tear-down
[=====] 1 test from 1 test suite ran. (0 ms total)
[ PASSED  ] 1 test.
Recording test results
Finished: SUCCESS
```

*Please contact TA if you has any trouble.*