Operating Systems
Project 01

- Simple User-level Unix Shell-

Specification

- This project is for the Linux, not xv6
- In this programming assignment, you will implement a simple user-level command line interpreter. (also called shell)
- Basically, shell operates in this way : when user types command to the prompt, shell creates a child process and execute the command.

```
$ ./shell

prompt> ls -al
(output of “ls -al” is shown here …)
prompt>
```

- Shell can be run in two ways : **Interactive mode & Batch mode**
- In interactive mode, shell displays a prompt and user types in a command at the prompt. (see above example)

Specification

- Each line may contain multiple commands separated with the ; character.
- Each command separated by ; character should be run simultaneously. It means that multiple processes run their command concurrently and parent process should wait all children before printing the next prompt. (As result, the output of this batched commands can be shown intermixed)
  - (ex) It is a valid command : **prompt> ls –al ; cat file ; pwd**
- In batch mode, shell is started by specifying a batch file on its command line. The batch file contains the list of commands that should be executed.
- In batch mode, you don't need to display a prompt, but should echo each line you read from the batch file back to the user before executing it.
- In both interactive and batch mode, shell stops accepting new commands when it sees the **quit** command or reaches the end of the input stream (i.e., the end of file or user types **Ctrl+D**)

Examples (Interactive mode)

```
$ ./shell
prompt>
prompt> ls
(ls output is shown …)
prompt> ls -al ; cat file
(outputs of two commands can be shown intermixed)
prompt> quit
$
```

Example (Batch mode)

```
$ vi [batchfile]

ls
/bin/ls -al
ls ; pwd ; cat file
```

```
$ ./shell [batchfile]

ls
(output of “ls”)
/bin/ls -al
(output of “/bin/ls -al”)
ls ; pwd ; cat file
(outputs of “ls”, “pwd”, “cat file”. this can be shown intermixed)
$
```

Useful Hints

- You should be familiar with **fork(), execvp(), and wait()/waitpid()** system call to implement this project. See the man page.
  - **In execvp(),** the list of arguments must be terminated with a NULL pointer;
- When reading/parsing the command from input string, you may want to look at **fgets(), strtok()** function.

Submission

- You should write defensive source code, your shell should print out error case thoroughly.
- Source code should be compiled with Makefile. (Source code that can’t be compiled with make is not graded.)
- You should upload your code to hanyang gitlab repository. (note that email submission is not accepted!!)

Preparing submission

1. Make a new directory named "**proj_shell**" in ~/xv6 directory, which is already managed by your Git

```
$ cd ~/xv6
$ mkdir proj_shell
```

2. You need to prepare 3 files into the proj_shell directory

   1. shell.c (your source file)
   2. Makefile (download this file from Piazza resource page)
   3. .gitignore (download this file from Piazza resource page)

```
$ ls -a ~/xv6/proj_shell
. .. .gitignore Makefile shell.c
```

Preparing submission

3. Build your project to confirm that it works well before submission

```
$ cd ~/xv6/proj_shell
$ make
gcc -g -Wall -c -o shell.o shell.c
gcc -o shell shell.o

$ ls -a
. .. .gitignore Makefile shell shell.c shell.o
```

Submission to the GitLab

4. Add your new project to the Git (now files are tracked by Git)

```
$ git add .
```

5. Commit files to the local repository

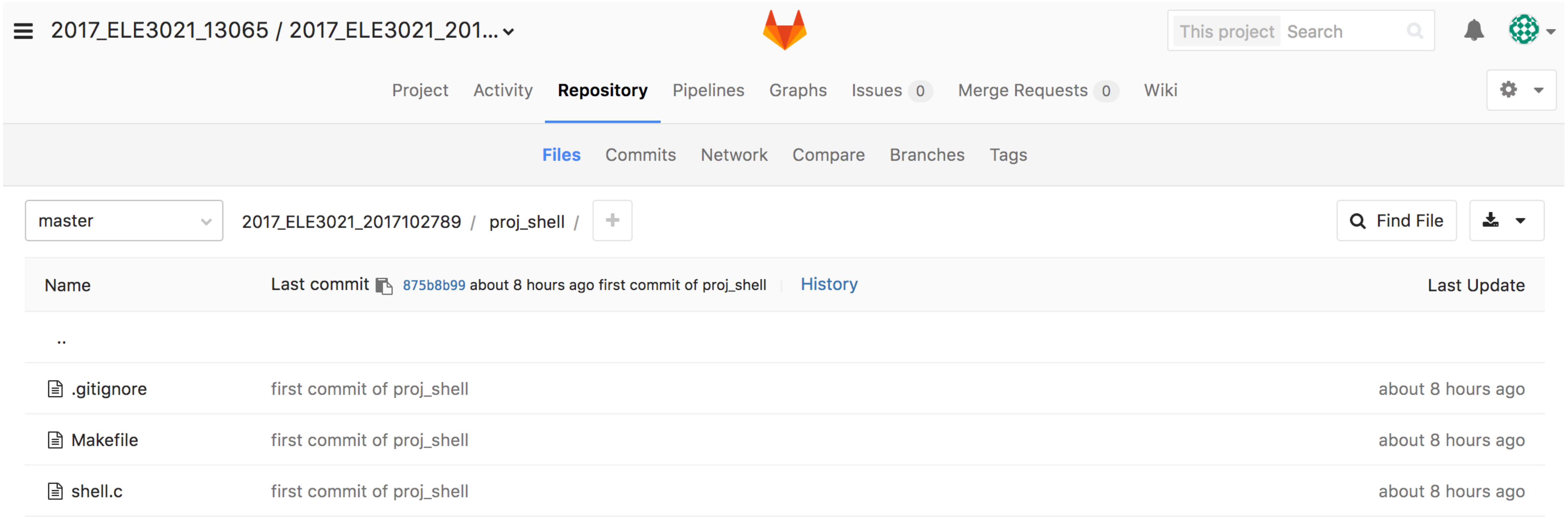
```
$ git commit –m “first commit of proj_shell”
```

6. Push local repository contents to the remote repository (Hanyang GitLab)

```
$ git push origin master
```

Submission to the GitLab

7. Now you can see your project files in the GitLab project page



Name	Last commit 875b8b99 about 8 hours ago first commit of proj_shell History	Last Update
..		
.gitignore	first commit of proj_shell	about 8 hours ago
Makefile	first commit of proj_shell	about 8 hours ago
shell.c	first commit of proj_shell	about 8 hours ago