

# Lab 3. Linux Tools for developers

## 1. Mục đích

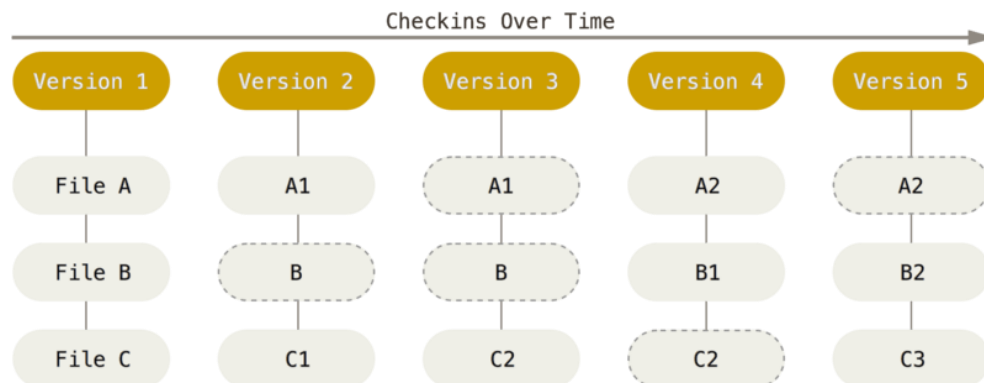
Thông qua bài thực hành này, sinh viên sẽ nắm vững các kiến thức sau:

- Sử dụng git để quản lý phiên bản source code xv6
- Sử dụng make để quản lý build source code xv6
- Sử dụng GDB để debug source code xv6

## 2. Tóm tắt lý thuyết

### 2.1. Git

Git là một chương trình quản lý phiên bản (distributed version control). Nó theo dõi và lưu trữ các phiên bản (version) của một tập hợp các file theo thời gian (ví dụ như các file source code của một project)



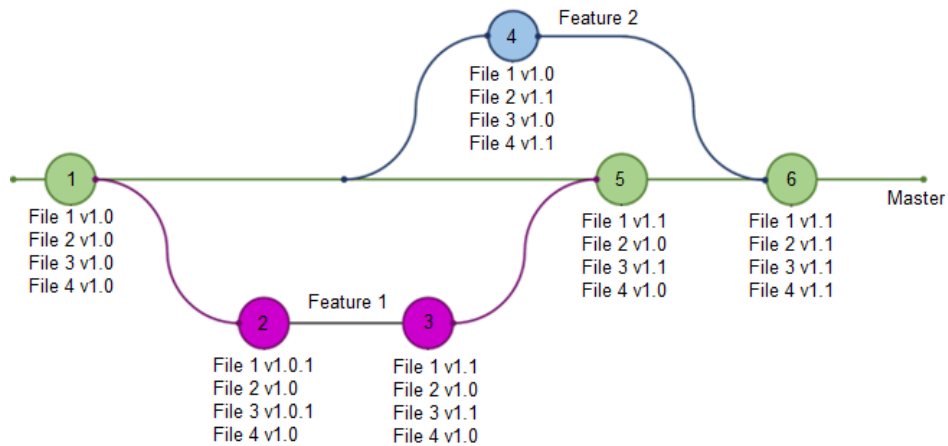
Mỗi khi người dùng thực thi lệnh commit, git lưu trữ trạng thái và nội dung của tất cả các file tại thời điểm đó (take snapshot) và tạo ra một **commit**

Mỗi commit được định danh bởi một chuỗi 40 kí tự hexa gọi là **commit hash** (SHA)



Nhiều commit có thể được phát triển từ một commit, tạo thành các nhánh rẽ gọi là các **branch**. Thông thường mỗi branch được dùng cho một mục đích nhất định.

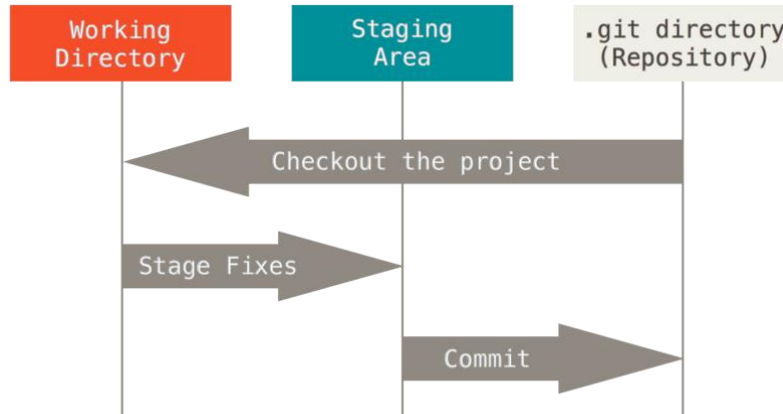
Ví dụ trong hình bên dưới, branch feature-1 và feature-2 đều bắt nguồn từ một commit, nhưng được dùng để phát triển hai tính năng độc lập không liên quan tới nhau. Tùy theo mục đích của người dùng, các branch có thể hợp nhất tại một commit thông qua quá trình **merge**



Git lưu trữ thông tin về các commit và các branch trong cơ sở dữ liệu (database) gọi là **repository (repo)**. Quá trình chọn một branch có trong repo để sử dụng gọi là **checkout**. Branch được chọn gọi là **checked-out branch/active branch/current branch**. Sau khi checkout, thư mục làm việc của người dùng (**working directory/working tree**) được khởi tạo bằng các file của checked-out branch

Trạng thái mỗi file trong working directory được theo dõi (tracking) bởi git như sau:

- Modified: file đã thay đổi
- Staged: file đã thay đổi. Trạng thái và nội dung của file đã được đánh dấu để chuẩn bị commit
- Committed: file đã được commit và lưu trữ trong repo



Bảng bên dưới là các câu lệnh git thường dùng. Sinh viên tham khảo thêm về cách sử dụng git tại [đây](#).

git status	Xem trạng thái hiện tại của working directory
git log --oneline	Xem lịch sử các commit của checked-out branch trong repo
git log --graph --oneline --all	Xem lịch sử các commit của tất cả các branch có trong repo dưới dạng đồ thị
git add --all	Stage tất cả các file có thể commit trong working directory
git commit -m <comment>	Commit các file đã stage kèm theo <comment>
git checkout <branch name>	Checkout branch <branch name> trong repo
git branch --all	Xem tất cả các branch có trong repo

git branch <new branch> <old branch>	Tạo branch mới <new branch> từ một branch cũ <old branch>
git clone <repo>	Tạo repo mới từ <repo>
git help <cmd name>	Xem hướng dẫn của <cmd name>. Ví dụ: git help commit

## 2.2. Make

make là chương trình quản lý build tự động. Nó tạo ra các file **target** từ các file **source** một cách đệ quy thông qua một tập hợp các **rule (recipe)** được khai báo trong **Makefile**.

Rule có dạng:

```
target : source(s)
[TAB]command
[TAB]command
...
```

Dòng đầu tiên khai báo sự phụ thuộc (**dependencies/prerequisites**) của target vào các file source. Các dòng tiếp theo là các command cần thực thi để tạo ra target từ các file source này. Target chỉ được build lại khi các file source của nó có sự thay đổi

Ví dụ 1: Bên dưới là một Makefile. Để build target myapp, gõ command **make myapp**

```
1  myapp: main.o f1.o f2.o f3.o
2      @echo Linking object files to create myapp
3      gcc main.o f1.o f2.o f3.o -o myapp
4
5  main.o: main.c main.h
6      @echo Compiling main.o from source code
7      gcc -c main.c -o main.o
8  f1.o: f1.c f1.h
9      @echo Compiling f1.o from source code
10     gcc -c f1.c -o f1.o
11  f2.o: f2.c f2.h
12     @echo Compiling f2.o from source code
13     gcc -c f2.c -o f2.o
14  f3.o: f3.c f3.h
15     @echo Compiling f3.o from source code
16     gcc -c f3.c -o f3.o
17
18  clean:
19     @echo Removing all object files and myapp
20     @rm -f *.o myapp
```

make cho phép định nghĩa các **variable** để sử dụng trong Makefile. Ví dụ 2:

```
1  OBJS = main.o f1.o f2.o f3.o
2  CC = gcc
```

```

3  myapp: $(OBJJS)
4      @echo Linking object files $(OBJJS) to create myapp
5      $(CC) $(OBJJS) -o myapp

```

Ngoài các variable do người dùng tự định nghĩa, một số **automatic variables** được định nghĩa bởi make như sau:

- `$$` Tất cả source của rule đang thực thi
- `$<` Source đầu tiên của rule đang thực thi
- `$@` target của rule đang thực thi

Ví dụ 3:

```

1  myapp: $(OBJJS)
2      @echo Linking object files $$ to create myapp
3      $(CC) $$ -o $@

```

Khi nhiều rule có cấu trúc giống nhau, chúng có thể được định nghĩa chung bằng cách sử dụng **pattern-matching rule**. Ví dụ 4:

```

1  CC = gcc
2
3  OBJJS = main.o
4  OBJJS += f1.o
5  OBJJS += f2.o
6  OBJJS += f3.o
7
8  myapp: $(OBJJS)
9      @echo Linking object files $$ to create myapp
10     $(CC) $$ -o $@
11
12 %.o: %.c %.h
13     @echo Compiling $$ from $<
14     $(CC) -c $< -o $$
15
16 clean:
17     @rm -f *.o myapp
18     @echo All object files and myapp have been removed

```

Sinh viên tham khảo thêm về cách sử dụng make tại [đây](#)

## 2.3. GDB

GDB là chương trình debugger có khả năng debug một chương trình ở mức source code và mức machine level. Sinh viên tham khảo thêm về cách sử dụng GDB ở [đây](#)

## 3. Thực hành

### 3.1. Git

Tạo thông tin tài khoản để định danh trong git. Trong đó EMAIL và NAME là riêng biệt cho mỗi sinh viên.

```
git config --global user.email "EMAIL"
```

```
git config --global user.name "NAME"
```

Ở thư mục home, clone git repo của source code xv6 về máy bằng command

```
git clone git://g.csail.mit.edu/xv6-labs-2022
```

```
fetel@ubuntufetel:~$ git clone git://g.csail.mit.edu/xv6-labs-2022
Cloning into 'xv6-labs-2022'...
remote: Enumerating objects: 7310, done.
remote: Counting objects: 100% (7310/7310), done.
remote: Compressing objects: 100% (3424/3424), done.
remote: Total 7310 (delta 3933), reused 7191 (delta 3870)/s
Receiving objects: 100% (7310/7310), 17.21 MiB | 1.39 MiB/s, done.
Resolving deltas: 100% (3933/3933), done.
fetel@ubuntufetel:~$
```

Chuyển thư mục làm việc vào thư mục source code của xv6 `xv6-labs-2022`. Liệt kê tất cả các branch có trong repo `git branch --all`

```
fetel@ubuntufetel:~/xv6-labs-2022$ git branch --all
* util
remotes/origin/HEAD -> origin/util
remotes/origin/cow
remotes/origin/pgtbl
remotes/origin/riscv
remotes/origin/syscall
remotes/origin/traps
remotes/origin/util
```

Tạo branch lab3 từ branch origin/util `git branch lab3 origin/util` và checkout lab3 `git checkout lab3`

```
fetel@ubuntufetel:~/xv6-labs-2022$ git branch lab3 origin/util
Branch 'lab3' set up to track remote branch 'util' from 'origin'.
fetel@ubuntufetel:~/xv6-labs-2022$ git checkout lab3
Switched to branch 'lab3'
Your branch is up to date with 'origin/util'.
```

Kiểm tra trạng thái của working directory bằng command `git status` và liệt kê tất cả các branch có trong repo `git branch --all`

```
fetel@ubuntufetel:~/xv6-labs-2022$ git status
On branch lab3
Your branch is up to date with 'origin/util'.

nothing to commit, working tree clean
fetel@ubuntufetel:~/xv6-labs-2022$ git branch --all
* lab3
util
remotes/origin/HEAD -> origin/util
remotes/origin/cow
remotes/origin/pgtbl
remotes/origin/riscv
remotes/origin/syscall
remotes/origin/traps
remotes/origin/util
```

Thay đổi source code của xv6 như sau:

1) Xóa file kernel/fs.h `rm kernel/fs.h`

2) Mở file kernel/fs.c bằng VSCode và xóa bỏ các include từ dòng 12 đến 22 và save file lại

3) Tạo file mytest echo THIS IS A NEW FILE > kernel/mytest

Các thay đổi trong thư mục working directory sẽ được tracking bởi git: `git status`

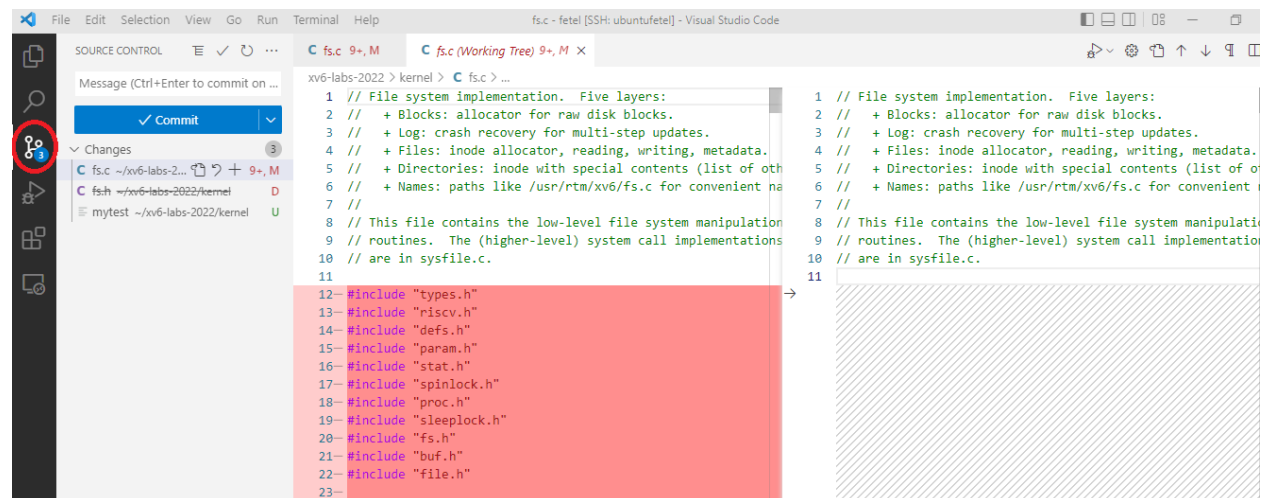
```
fetel@ubuntufetel:~/xv6-labs-2022$ git status
On branch lab3
Your branch is up to date with 'origin/util'.
```

```
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   kernel/fs.c
        deleted:    kernel/fs.h
```

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        kernel/mytest
```

no changes added to commit (use "git add" and/or "git commit -a")

Xem chi tiết về sự thay đổi của từng file trên VSCode, trong mục Source Control



Stage tất cả các file và commit: `git add --all; git commit -m "my first experience with git"`

```
fetel@ubuntufetel:~/xv6-labs-2022$ git commit -m "my first experience with git"
[lab3 060d520] my first experience with git
3 files changed, 1 insertion(+), 72 deletions(-)
delete mode 100644 kernel/fs.h
create mode 100644 kernel/mytest
```

Kiểm tra lại lịch sử các commit của checked-out branch lab3 trong repo `git log --oneline`

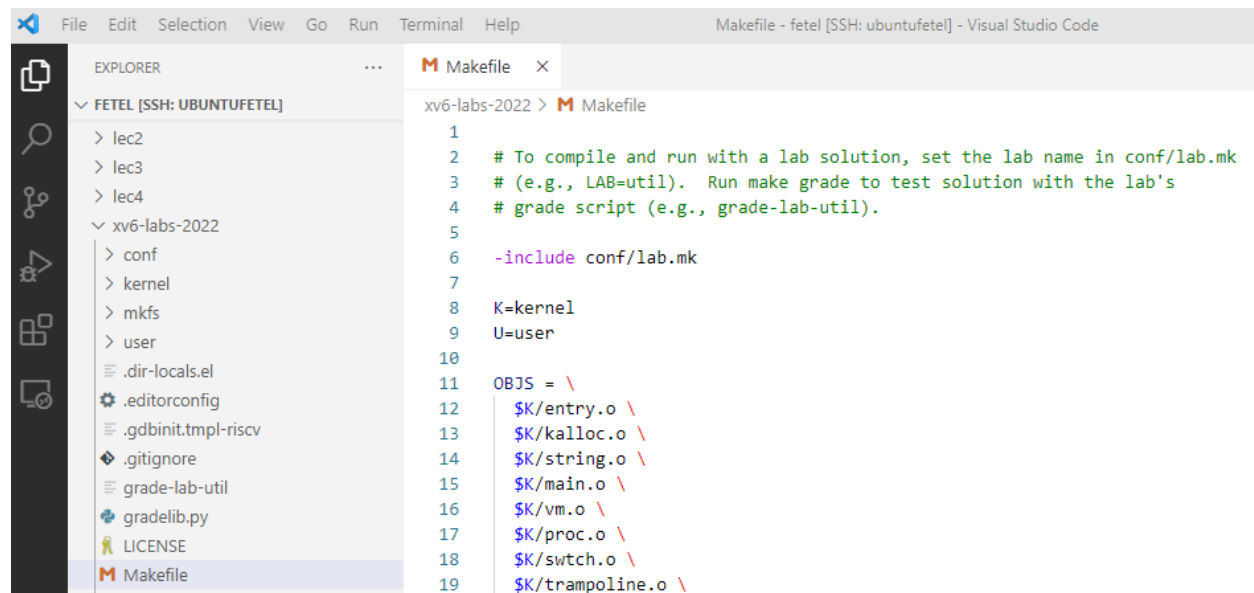
```
060d520 (HEAD -> lab3) my first experience with git
dc9153f (origin/util, origin/HEAD, util) Fix year
4d74938 2022 submission site
0c477a6 Merge branch 'riscv' into util
3d6ce9b Separate tests in slow and quick. The slow tests run xv6 out of memory, out of disk space, or test big directories.
ed101be comment the sfences
581bc4c sfence before enabling paging
29ce316 Merge branch 'riscv' of g.csail.mit.edu:xv6-dev into riscv
```

Các thay đổi đã thực hiện với file `fs.h`, `fs.c` và `mytest` nằm ở commit trong branch `lab3`. Do đó khi checkout một branch khác (Ví dụ: `git checkout util`), các thay đổi đó sẽ không hiện hữu.

### 3.2. Make

Source code của `xv6` được build bởi `gcc` và quản lý build bởi `make`.

Chuyển thư mục làm việc vào thư mục source code của `xv6`. Checkout branch `util` `git checkout util`, và mở file `Makefile` trên VSCode.



Bảng bên dưới liệt kê các target chính có trong Makefile:

<code>\${K}/kernel</code>	kernel của <code>xv6</code>
<code>mkfs/mkfs</code>	chương trình dùng để build file system cho <code>xv6</code>
<code>fs.img</code>	file system của <code>xv6</code>
<code>qemu</code>	Chạy chương trình <code>qemu</code> emulator cho kiến trúc RISC-V multi-core, load và chạy <code>xv6</code> kernel với file system <code>fs.img</code>
<code>qemu-gdb</code>	Tương tự như target <code>qemu</code> nhưng chạy emulator để debug với GDB

Thực thi command `make qemu`. Sau khi build xong, `qemu` khởi động `xv6` thành công và bắt đầu phiên làm việc như hình bên dưới

```

xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ 

```

Các command từ giờ trở đi sẽ được thực thi trên `xv6`. Khác với `ubuntu`, chương trình shell của `xv6` rất đơn giản và thiếu nhiều tính năng. Gõ command `ls` để liệt kê các file và thư mục trong thư mục hiện tại. Hầu hết các file được liệt kê là chương trình có thể được thực thi trên shell. Lệnh `ls` vừa gõ là một trong số đó

```
$ ls
.          1 1 1024
..         1 1 1024
README    2 2 2227
xargstest.sh 2 3 93
cat        2 4 32328
echo       2 5 31200
forktest   2 6 15304
grep       2 7 35680
init       2 8 31992
kill       2 9 31200
ln         2 10 31112
ls         2 11 34232
mkdir      2 12 31240
rm         2 13 31224
sh         2 14 53472
stressfs   2 15 32096
usertests  2 16 180584
grind      2 17 47296
wc         2 18 33320
zombie     2 19 30768
console    3 20 0
$
```

Để kết thúc phiên làm việc trên xv6, nhấn Ctrl + a, sau đó nhấn x. Terminal sẽ trở lại kết nối với ubuntu

### 3.3. GDB

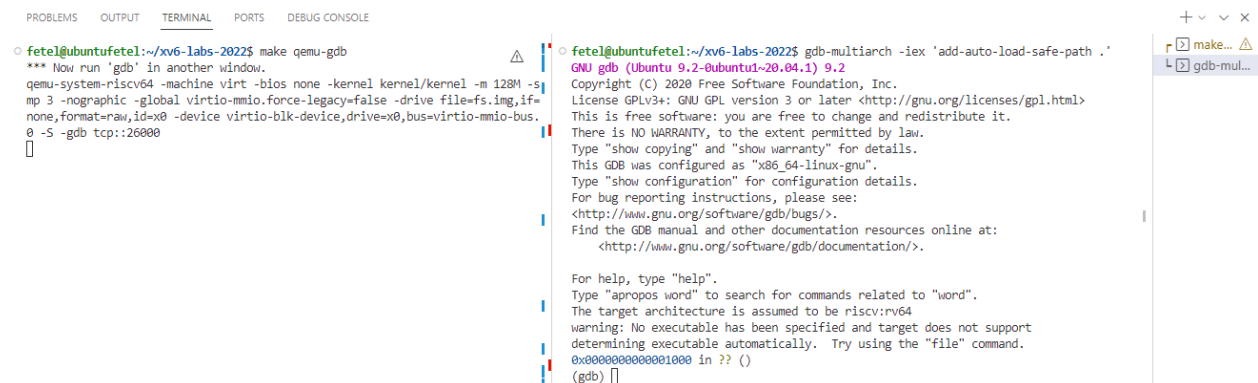
Trong hầu hết các trường hợp, sử dụng hàm print là đủ để debug xv6.

GDB thường được dùng khi cần đến các tính năng debug nâng cao như chạy từng dòng lệnh hay giám sát các variable. Để debug với GDB, mở 2 cửa sổ Terminal trên VSCode



Terminal 1 thực thi command `make qemu-gdb`

Terminal 2 thực thi command `gdb-multiarch -iex 'add-auto-load-safe-path .'`



Phiên Debug đã sẵn sàng. Nhập các lệnh GDB vào Terminal 2 để tiến hành debug.

Ví dụ:



1	help	Hiển thị trợ giúp
2	load kernel/kernel	Load kernel image
3	info thread	In ra thông tin các CPU thread. Trong bài lab này, RISC-V có 3 CPU
4	break start	Đặt breakpoint tại hàm start cho tất cả các CPU
5	break main.c:20 thread 1	Đặt breakpoint tại dòng 20 của file main.c cho CPU1
6	info break	In ra danh sách các breakpoint
7	delete breakpoints 1	Xóa breakpoint đầu tiên trong danh sách
8	continue	Tiếp tục thực thi cho đến khi bất kì CPU nào gặp breakpoint
9	frame	In ra dòng source code sẽ thực thi tiếp theo (của CPU1)
10	list	In ra các dòng source code gần vị trí hiện tại
11	info registers	In ra registers của CPU hiện tại (CPU1)
12	next	Thực thi dòng code tiếp theo
13	step	Giống với next, nhưng khi gặp hàm con thì chuyển vào hàm con
14	thread 2	Chuyển sang CPU2
15	frame	In ra dòng source code sẽ thực thi tiếp theo (của CPU2)
16	print started	In ra giá trị của biến started
17	info registers	In ra registers của CPU hiện tại (CPU2)
18	where	In ra vị trí hiện tại trong call stack (của CPU2)
19	quit	Kết thúc debug

## **BÁO CÁO THỰC HÀNH**

Sinh viên: .....

MSSV: ..... Nhóm: .....

### **Bài 1:**

File .gitignore trong working directory có nhiệm vụ gì? Làm thế nào để hủy bỏ các thay đổi đang thực hiện trong working directory?

### **Bài 2:**

Xác định các rule được dùng để build các target chính trong Makefile của xv6. Cho biết các target đó phụ thuộc vào các file nào trong source code của xv6