ITF22519: Introduction to Operating Systems

Fall Semester, 2021

Lab1: Introduction to Linux and GitHub

Submission Deadline: September 2nd, 2021 23:59

"Practice makes perfect!"

This lab will help you to be familiar with the basic use of Linux operating system and GitHub. Please make sure that you are comfortable using all Linux command lines in this document. You are also encouraged to ask questions if there is anything you do not understand because future labs will be based on this material.

In this lab, you are expected to go through the entire material and do few exercises. However, if you feel you are already a Linux expert, you can directly go to the `Exercises` section.

1 Introduction to Linux

1.1 Logging in

Linux is a multi-user operating system. It allows multiple users to have accounts and to be logged in at the same time. Once you feel marginally comfortable with the desktop, open a terminal by right-clicking on the desktop and selecting Open Terminal from the popup menu. If you already have one Linux distribution installed on your computer, you can use your computer for all the labs of this course. Otherwise, we provide a Linux server at itstud.hiof.no to which you can connect via SSH protocol. You have to use your HiOF username and password to connect to this server. To do so, on Windows 10 or Mac machines, try the following command on your command prompt:

```
$ ssh your_username@itstud.hiof.no
```

or

```
$ ssh your_username@itstud
```

Note: If SSH clident is not installed on your Windows 10 machine yet, follow this link: SSH to install OpenSSH client. Otherwise, you can download and try PuTTY software at: PUTTY. To use the server at home, you need also connect to HIOF VPN. Check this link: VPN Link.

1.2 Command Line

The command line in UNIX is actually another program, called a *shell*. A shell is a computer program that presents a command line interface (CLI) which allows you to control your computer using commands entered with a keyboard instead of controlling graphical user interfaces (GUIs) with a mouse/keyboard combination. The shell allows you to run other programs or scripts, and is generally useful for managing your computer once you know which commands to use. Most Linux distributions ship with *bash* as the default shell. This shouldn't be confused with the terminal emulator, which simply presents a GUI for command-line programs and happens to run bash by default.

When you start a terminal, the first thing you will see is the *prompt* (often called the command line). As its name would suggest, it is prompting you to enter a command. It should look something like this:

```
[username@host currentdirectory]$
```

Type the commands after the prompt $. Let's practice with some simple commands in Linux. First, check who are currently logged in the computer by using the command `who`

```
$ who
```

You should see the information about current users together with the processes they are using and time/date they logged into the system. Note that all Linux commands can have several *options* and *arguments* that may change the way they work. For example, the command `who` has many options among which the option `-q` shows all login names and number of users logged on, option `-b` shows the time of the last system boot, option `--version` prints out version information and exit. Practice with those options by typing the following commands and see how they work:

```
$ who -q
$ who -b
$ who --version
```

If you want a quick view of the calendar, type:

```
$ cal
```

By default, this command shows the current month calendar as output. If you want to print out a calendar for a specific year and month, use this syntax: `cal month year`. Do you want to print out the month and the year you were born? If you want to show the calendar of this year, type:

```
$ cal 2021
```

In Linux, the command `echo` is used to display a line of text to the screen or terminal window. We will look into it a bit more in Lab 10 with Bash scripting. Now, let's do a simple practice with `echo` command:

```
$ echo Welcome to the Labs!
```

Then, you would see that the line `Welcome to the Labs!` is displayed in the screen.

The Linux shell has a number of built-in system variables, which store text strings and are identified by variable names. You can look at the contents of these by putting a $ (dollar sign) in front of the variable name. Type the following commands and see how they work:

```
$ echo Hello $USER
$ echo $HOSTNAME
$ echo $TERM
$ echo $PATH
```

Linux also provides you with commands to check hardware and system information. For example, the command `lscpu` reports information about the cpu and processing units. It does not have any further options or functionality. To practice, type:

```
$ lscpu
```

To check all the peripheral component interconnect (PCI) buses and details about the devices connected to them, type:

```
$ lspci
```

Practice yourself with the following commands:

```
$ cat /proc/version
$ cat /proc/partitions
$ cat /proc/cpuinfo
$ df
$ lsblk
$ free -m
```

Now, make a directory to save your work for this course in a folder called `Introduction2OS`:

```
$ mkdir Introduction2OS
```

Then, change into that directory using:

```
$ cd Introduction2OS
```

Next, create a new file and edit it with *Nano* text editor:

```
$nano newfile.txt
```

You will then enter Nano editor interface. Enter some text and press `Ctrl-X` to save the file and exit Nano. Back on the command line, type the following:

```
$ ls
```

The file `newfile.txt` should be listed.

1.3 Current directory and Parent directory

Type:

```
$ ls -a
```

The command `ls -a` lists all contents of a directory. To show flie sizes in a human readable format, use the following command:

```
$ ls -a -l -h
```

or

```
$ ls -alh
```

You should see two additional entries – "`.`" and "`..`". The single dot (`.`) is a reference to the current directory, and the two dots (`..`) is a reference to the parent directory. To change to the parent directory, type:

```
$ cd ..
```

```
$ ls
```

You should see the `Introduction2OS` directory in the list since you're now in the parent directory.

1.4 Absolute path and Relative path

You need to know two more things about directories. They are *absolute* and *relative* paths.

An *absolute path* is a path that starts with "/" or "∼". Because it starts with the root of the file system, it always referto the same file no matter what your current directory is.

A *relative path* is a path that starts with any other character. It is relative to the current directory, and will refer to different files depending on the current directory. Type `pwd` to see the current absolute path. You should see something like `/home/username`, which indicates you're in the `username` directory under the `home` directory. This is where to find your account in Linux. It is the default current directory when you login. You can always go back to it by typing:

```
$ cd
```

Now try using an absolute path to switch directories. Type (replacing username with your own username):

```
$ cd /home/username/Introduction2OS
```

You should now be in your Introduction2OS directory. To confirm this, type:

```
$ pwd
```

To return to your home directory, type:

```
$ cd
```

Now switch to the directory again using a relative path:

```
$ cd Introduction2OS
```

Use `pwd` to verify that you are now in the subdirectory. So far, there is no difference between those two ways to get to a directory. However, suppose that you were in another user's home directory. In that case, `cd Introduction2OS` command would take you to his or her `Introduction2OS` directory. But, if you used the absolute path with your username in the path, your would go to your `Introduction2OS` directory. This difference is not all that interesting now, but be aware of it for future use. There will be occasions when you will want to specify the file or directory you want to work with unambiguously by using an absolute path. You can use relative and absolute paths to describe files and directories in many commands. For example, if you're in your home directory, the following two commands should be equivalent:

```
$ ls /
$ ls ../..
```

The reason is that the first command uses an absolute path (it starts with /), then it will always return the same result no matter where you are. If you're in the directory `/home/username`, then `../..` resolves to the parent of the parent of the current directory, or /. If you were in `/home/username/Introduction2OS` and typed `ls ../..`, you would see the directory listing for `/home` instead.

A few more general tips:

- Bash has tab-complete feature which allows you to type the

  rst few characters and press tab to auto

  ll in the rest. Pressing tab twice will show all possible values what can be entered.

- You can use the arrow keys to look through recently typed commands. In bash, `Ctrl-R` will search your command history for the text you type.

- `itstud` is a file server. To access your remore files on a Windows machine, try Filserver-Windows. On Mac, try Filserver-Mac. Otherwise, SFTP clicents such as FileZilla can also be used to transfer and access your files on a Windows or a Mac machine.

Recommended Editors: If you are not already proficient with a Unix editor, you should pick one to become proficient with. Here are some text editors that you may choose to use throughout this class: `gedit, gvim, kwrite, vi, vim, nano, emacs, and xemacs`.

1.5 Manual pages

Manual pages are your best friends in Unix; they provide information you can digest on almost all commands, applications, system calls, programming languages, etc. For example, to view the manual page for the ping command, type `man ping` in your terminal. You will see something like this:

```
PING(8)              System Manager's Manual: iputils                PING(8)

NAME
ping, ping6 - send ICMP ECHO_REQUEST to network hosts

SYNOPSIS
ping [ -LRUbdfnqrvVaAB] [ -c count] [ -i interval] [ -l preload] [
-p pattern] [ -s packetsize] [ -t ttl] [ -w deadline] [ -F flowlabel]
[ -I interface] [ -M hint] [ -Q tos] [ -S sndbuf] [ -T
timestamp option] [ -W timeout] [ hop ...] destination

DESCRIPTION
ping uses the ICMP protocol's mandatory ECHO_REQUEST datagram to elicit an ICMP
ECHO_RESPONSE from a host or gateway. ECHO_REQUEST datagrams ("pings") have an
IP and ICMP header, followed by a struct timeval and then an arbitrary number of
"pad" bytes used to fill out the packet.
```

There may also be more help in info pages. The compiler's manual can be reached using `info gcc`. When they exist, the info pages for a given command may contain more information about what the program does rather than simply a description of its syntax.

Manpages are divided into several sections; notice `PING(8)` in the above example means ping is in section 8. A description of the different sections is in `man man`. Some important sections are: system calls are in section 2, library functions are in section 3, system programs are in section 1 (or 8 for administrative tools). For example, `man read` will not show the documentation for the system call `read()`; you must run `man 2 read` to get the correct entry.

In many of the future labs, you will be expected to look up important information in man pages. Take some time now to look up a few things. Try looking up the man page for "ls" and "signal". When you're finished with an info or man page, just type q to quit.

1.6 UNIX tutorial

In order for you to get familar with UNIX Operating Sytem, we would recommend you to go through the first FIVE tutorial on this page: Unix.

2 Exercices

2.1 Exercise 1 (40 pts)

Explain what following commands do:

```
cat file1 file2 file3 | grep cat
who | wc -l > file1.txt
sort < file1.txt > file2.txt
ps | wc -l
```

2.2 Exercise 2 (20 pts)

Which command (and its options) used to completely remove a directory which has some sub-directories and files inside?

2.3 Exercise 3 (40 pts)

1. What does each of the following commands do:
   - rmdir
   - mv
   - cp
   - less
   - pushd
   - popd
2. What would you type in your termial to learn more about command `ls` ?

3 Git Subversioning

All labs in this course will use the git subversioning system. In general, Git allows multiple people to work on the same code base at once, each with their own entire version of the source tree.

We will you GitHub to manage all of the assignments. If you do not already have a GitHub account please go to http://www.github.com and create a free account now. Once you created your Github account, please make sure to associate it with your school email. Also, please update your profile with your full name in order to help us to identify your submissions easily.

3.1 Create SSH Keys

GitHub uses a Secure Shell (SSH) connection to push and pull the Git repository to and from the cloud. A SSH is a cryptographic network protocol for operating network services securely over an unsecured network [Wikipedia]. Typical applications include remote command line, login, and remote command execution, but any network service can be secured with SSH. Public/Private keys are used to secure the connection between your local computer and GitHub. Now, check to see if SSH keys already exist by running on the terminal.

```
$ ls -al ~/.ssh
```

If you see a file listed called `id_rsa.pub`, then skip the Install SSH Keys section. Otherwise, please continue to create new keys. Run the following command in the terminal replacing the email address with your email address. If the command asks for your input (e.g., name of the key file, passphase, etc.), do not enter anything, just press enter:

```
$ ssh-keygen -t rsa -C "your_email@hiof.no"
```

3.2 Install SSH Keys

Once you have generated SSH keys, you must install the public key to github. Open the public key in a text editor or try:

```
$ cat ~/.ssh/id_rsa.pub
```

Then, copy all of the contents of the file into the clipboard (by select all the content and right click). Next, in a web browser, log into GitHub and click the **Settings** under your account button in the upper right corner. After that, click **SSH and GPG keys** option. Click **New SSH** key and enter **NewHIOF Key** as the title of the key. Next paste the contents of the key file into **NewHIOF Key** and click Add SSH key. Next we will test the key. Enter the following command in the terminal:

```
$ ssh -T git@github.com
```

If successful, you should be asked if you want to continue. Type *yes* and press *Enter*. If you setup a passphase you will now be asked to enter the passphase. If everything works, you should get a welcome message from Github: `Hi GithubUsername! You've successfully authenticated, but GitHub does not provide shell access.` If you get an error, please ask us for assistance.

Note: If you would like to do development work on another computer, you will need to follow these steps again giving the key another different title.

3.3 Access Class GitHub

To access the class GitHub, you need to be granted access right. Please ask us to add you into the group. Next, you need to accept the class assignment which . (We already provided you with a link to the assingment. Check your email or you can also find its Annocement in Canvas). Follow the steps in the link and you will be assigned with a private repository named `labs-github_username`. Only you, lecturers, and teaching assistances (TAs) have access to your repository. For this lab and future labs, you should use this repository to access the labs instruction or mandatory assignments and submit your lab reports and exercises.

You can now clone the master labs repository. First, go to the directory you created ealier where you will save your files for this class. Use the following commands to get the labs master repository:

```
$ git clone git@github.com:Introduction2OS/labs.git
$ cd labs
```

By default, the remote called origin is set to the location that you cloned the repository from. You should see the following:

```
$ git remote -v
origin git@github.com:Introduction2OS/labs.git (fetch)
origin git@github.com:Introduction2OS/labs.git (push)
```

We want to change the remote to your personal repository so you can submit this and future labs. To do that, enter the following commands replacing with your Github username:

```
$ git remote rename origin upstream
$ git remote add origin git@github.com:Introduction2OS/labs-github_username.git
```

If everything is set up correctly you should get the following:

```
$ git remote -v
origin git@github.com:Introduction2OS/labs-github_username.git (fetch)
origin git@github.com:Introduction2OS/labs-github_username.git (push)
upstream git@github.com:Introduction2OS/labs.git (fetch)
upstream git@github.com:Introduction2OS/labs.git (push)
```

Finally let's initialize your private repository by issuing the following command:

```
$ git push -u origin +master
```

3.4 Using The Class GitHub

For all future labs, you will get the code and other materials by issuing the following commands in your labs directory:

```
$ git pull upstream master
```

Come to your lab1 directory:

```
$ cd lab1
```

Create a new text file with Nano or any other editors you like:

```
$ nano newfile.txt
```

Enter some text and save the file. Now go back to the labs directory:

```
$ cd ..
```

To displays the state of your working directory in the lab directory run the command:

```
$ git status
```

This command will show which files have changed or been created. Add the lab you want to submit by typing:

```
$ git add lab1/
```

Next, run git status again and make sure all the
files that should be submitted show up. Then, commit your changes to your local repository with a meaningful commit message using:

```
$ git commit -m "This is my first commit"
```

Now that you have your local repository up-to-date, you need to push your changes to the github. To do this, run the command:

```
$ git push
```

Finally, check the GitHub website to make sure your changes show up correctly. Note, you can (and should) commit your changes often and push as often as you like. We will only grade your last push, and by committing often you have the ability to roll back changes or recover files if something happens.

4 What To Submit

Complete the exercises in this lab. Put your **lab1_report.txt** file inside the lab1 directory of your repository. Run `git add` and `git status` to ensure the file has been added and commit the changes by running `git commit -m "Your Commit Message"`. Finally, submit your files to GitHub by running `git push`. Check the GitHub website to make sure that all files have been submitted.