ECE482 — Introduction to Operating Systems

Lab 1

Manuel — UM-JI (Fall 2022)

Goals of the lab

- Recognize the common computer hardware
- Initial git setup
- Review on Shell

ECE4820 Tasks

1 Hardware overview

Some components might be missing, in such a case figure out their expected location.

In the computer locate:

- The motherboard
- A Hard Disk Drive
- An Optical disk drive
- The PC power supply
- A PCI card

On the motherboard locate:

- The RAM
- A SATA socket
- A PCI/PCI-e slot
- The CPU
- The North and South bridges
- The battery
- The BIOS

Answer the following questions:

- Where is the CPU hidden, and why?
- What are the North and South bridges?
- How are the North and South bridges connected together?
- What is the BIOS?
- Take out the CPU, rotate it and try to plug it back in a different position, is that working?
- Explain what overclocking is?
- What are pins on a PCI/PCI-e card and what are they used for?
- Before PCI-e became a common standard many graphics cards were using Accelerated Graphics Port (AGP), explain why.

2 Gitea usage

Register on our git server at `http://focs.ji.sjtu.edu.cn/git`. We will use Gitea all along the semester. For the various group projects student **must use git**: part of their grade will be based on their commits.

Please closely follow the TAs' instructions and ensure your are fully setup for the rest of the semester. In particular by the end of the lab **you should have uploaded your ssh public key** on Gitea.

2.1 Git usage

Basic git usage will be studied in a future lab. In the meantime get familiar with:

- Opening and using issues;
- Simple commands such as `clone`, `commit`, `push`, and `pull`;
- Using the Wiki and markdown, they will be used for the projects documentation;

2.2 Agile development with Gitea

In ECE482 we expect you follow the adgile development strategy which is adopted by most companies nowadays. If you have any question of need further explanations feel free to ask. Here is a brief summary of the expected Gitea workflow in the course.

Generic agile development requirements:

- For each project create a meaningful project board and update it on a regular basis; Common boards from left to right: Backlog, Todo, In progress, In review, Done;
- Properly set the milestones and attach issues to them;
- Use the provided feature issue to open an issue for each task;
- Always have your issues assigned to someone;
- For each issue, the timer is started and stopped when it is worked on;
- After working on an issue a brief summary of the progress is logged as a comment in the issue;
- All git commits are atomic;[1]
- All git commits message follow conventional commits standard;[2]
- Features are developed in individual branches, and each branch is named following the following standard `feat/name`, `fix/name`, etc. depending what is done in the branch;
- Pull requests clearly explain changes that are applied and each reviewer properly reviews the code, test the changes, and provides insightful feedback;
- The software is documented in wiki pages. Also create a Roadmap page summarizing the expected development process (listing the tasks by milestone and showing which ones have been completed). In the Roadmap, include a rough time estimate for each task.
- For team works group meetings are held twice a week and logged in an issue to report the progress;[3]

If you need any clarification on our requirements please ask us at any stage during the course.

[1] Commits are small, i.e. do a single thing. Although their is no official bound on the number of lines, typically an atomic commit is seldom more than 30 lines long.

[2] Read through the conventional commits specifications, then install and setup Commitizen. Commitizen is a small tool that will ensure your commits comply with conventional commits specifications.

[3] Such meetings are known as *scrum meetings*. They should not last more than 10 min.

3 Command line interface

The goal of this part of the lab is to get use of the terminal and command line interface. The command line interface is often faster that graphical user interfaces as there is no need to move the mouse around to click in many "random places" on the desktop; commands are usually simple and well documented.

We start by briefly explaining how to perform basic tasks and then move on to writing simple *shell scripts*. A shell script is a program meant to be run by the command line *interpreter*; it often consists in a list of command lines together with some loops and conditional statements, everything being stored in a file with a `.sh` suffix.

Scripts have many usages and can really simply life when facing repetitive tedious tasks. Common example of such tasks are automatically updating and generating slides every semester, automatically uploading documents and setting up assignments on Canvas, or even filling in the Honour Council form in case of cheating suspicions.

Knowing the basics on how to use the command line interface to write simple scripts can highly simplify your life all along your studies and even in your future career.

3.1 Basic Unix commands

Simple commands to run basic tasks:

- Directory tree navigation: `ls, cd`
- File and directory manipulation: `mv, rm, cp, mkdir, rmdir`
- File reading: `cat, head, tail`
- Text manipulation: `echo, cut`
- Search: `find, grep`
- Navigate through the command history: up and down keys, ctrl-r

To access the manual on each of the above command use `man`, e.g. `man ls`. To navigate in the documentation use the direction keys or search for some keyword, e.g. once in the `ls` manual, type `/author`. You will then jump to the first occurrence of the word "author". Press `n` to jump to the next one and `N` for the previous one.

Some manual pages can be very very long (several thousands of lines), so do not waste time reading from the beginning to the end. Instead use keywords to search or scan through it. An online search can also help pinning down the correct parameters.

Various shells exist with more of less features and slightly different syntax. Being familiar with one should allow you to be comfortable with others. The most common ones are `dash` (basic shell), `bash` and `zsh` which both have many features. Most Linux distributions use bash as default and macOS recently moved from bash to zsh.

For the sake of simplicity in the following guidelines we focus on bash, but all the tasks can be completed using zsh or dash if you prefer.

3.2 Shell scripting

Some basic shells such as `mumsh`, the shell developed in VE482, are simple command line interfaces without any real programming features. All the previously mentioned, dash, bash, and zsh, in fact implement a fully features programming language best used for writing scripts. The following information remain very

basic and minimalistic, numerous other powerful features being available. Feel free to poke around the man page to learn more or direct your attention to other important topics such as *regular expressions*, or `sed` and `awk`.

To run a shell script open a terminal and simply type `interpreter scriptname`, where `interpreter` is the name of your shell, e.g. bash or zsh, and `scriptname` is the name of your script (often ending with `.sh`).

3.2.1 Basics

```
#!/bin/bash
# the first line of a bash script is always #!/bin/bash -- use #!/bin/zsh for zsh
# anything following # is a comment
a=asd # assign asd to variable a
echo $a #display the content of variable a
echo $1 #first argument to the script
echo $2 # second argument to the script, more arguments $3, $4...
echo $@ # all the arguments
echo $? # exit code from the previous command
# refer to bash man page for more advanced operations on variables e.g.
a="123 4.jpg";
echo ${a%.jpg} ${a:2:3} # extract partial content from variable a
echo ${#a} # get the length of variable a
```

3.2.2 Conditional statements

```
[ expression = value ] # test an expression, man test for more details

# if statement
if [ $a =  "qwe" ] ; then
  list of statements
fi

# case keyword
case $i in
  a) list of statements
  ;;
  b) list of statements
  ;;
  *) list of statements #default actions
esac
```

3.2.3 Loops

```
# for loops (list is a space separated list of elements (e.g. filenames))
for i in list ; do
  list of statements
done

# for loops (iterate a predefined number of times)
for((i=0; i<10; i++)) ; do
  list of statements
done

# while loops
while some expression ; do
  list of statements
done
```

3.2.4 Arrays

```
# simple array
a[3]=4; echo ${a[3]}
i=2; a[$i]=1

# associative array
declare -A b=([key1]=value1 [key 2]=value2);
echo ${b[key1]}; echo ${b[key2]}; echo ${b[key 2]};
c=key1; echo ${b[$c]}
```

3.2.5 Functions

```
name () {
  core of the fuction
}
```

3.3 Tasks

After this brief introduction lets play with the shell!

Answer the following questions, only refering to man pages:

- Use the `mkdir`, `touch`, `mv`, `cp`, and `ls` commands to:
  - Create a file named `test`.
  - Move `test` to `dir/test.txt`, where `dir` is a new directory.
  - Copy `dir/test.txt` to `dir/test_copy.txt`.
  - List all the files contained in `dir`.
- Use the `grep` command to:

  - List all the files form `/etc` containing the pattern 127.0.0.1.
  - Only print the lines containing your username and root in the file `/etc/passwd` (only one grep should be used)
- Use the `find` command to:
  - List all the files from `/etc` that have been accessed less than 24 hours ago.
  - List all the files from `/etc` whose name contains the pattern "netw".
- In the `bash` man-page read the part related to redirections. Explain the following operators `>`, `>>`, `<<<`, `>&1`, and `2>&1 >`. What is the use of the `tee` command.
- Explain the behaviour of the `xargs` command and of the | operator.
- What are the `head` and `tail` commands? How to "live display" a file as new lines are appended?
- How to monitor the system using `ps, top, free, vmstat`?
- What are the main differences between `sh`, `bash`, `csh`, and `zsh`?
- What is the meaning of `$0, $1,…, $?, $!`?
- What is the use of the `PS3` variable? Provide a short code example.
- What is the purpose of the `iconv` command, and why is it useful?
- Given a variable `$temp` what is the effect of `${#temp}, ${temp%%word}, ${temp/pattern/string}`.
- Search online (not in the man pages), how files are organised on a Unix like system. In particular explain what are the following directories used for:
  - `/`
  - `/lib`
  - `/usr/lib`
  - `/srv`
  - `/sbin`
  - `/bin`
  - `/mnt`
  - `/usr/src`
  - `/media`
  - `/dev`
  - `/boot`
  - `/usr/bin`
  - `/proc`
  - `/opt`
  - `/vmlinuz`
  - `/etc`
  - `/usr/share`
  - `/sys`
  - `/var`
  - `/initrd.img`

Now lets write some real scripts!

Write a game where the computer selects a random number, prompts the user for a number, compares it to its number and displays "Larger" or "Smaller" to the user, until the player discovers the random number initially chosen by the computer.

Hints: the following commands might be helpful.

- `echo $RANDOM $((10%3))`
- `man read`
- To search in a man page some characters need to be *escaped*, e.g. to search for `$((`, do `/\$\(\(`
- Above commands might have to be adjusted to meet the requirements and avoid bugs