VG100
Introduction to Engineering

Homework 1

Michele & Manuel — UM-JI (Summer 2020)

- Download the homework support materials
- Always include comments in the code
- Before starting think of the program structure
- Keep testing and improving your code
- Write a single `README` file per assignment
- When altering code list your changes in a `changelog`
- Archive the files (`*.{zip|tar}`) and upload on Canvas

The goal of this first homework 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.

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.

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`).

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
```

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
```

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
```

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]}
```

2.5 Functions

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

3 Exercises

After this brief introduction lets play and program games!

Ex. 1 — *Shell commands*

Ever dreamt of visiting a dungeon, fighting monsters, and collecting treasures? Or even better, doing it using bash commands? Get a copy of Bashcrawl[1] and let you dream become true!

[1] Available in the homework git repository.

Ex. 2 — *Guess!*

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

Ex. 3 — *Hangman*

Write a hangman game where a word is randomly chosen among all the first words of each line in the `cmd.txt`[1] file. The user can make up to six mistakes before getting hanged. Whether winning or loosing the user should be presented with the short description of the command, i.e. the second part of the line, and prompted to know whether the manpage should be opened or not.

Hints: the following tips might be helpful.

- `sed -n "$(($RANDOM%20))p" cmd.txt`
- `cut -d ' ' -f 1`
- `sed 's/\([^ ]\+\)[[:space:]]*\(.*\)/\1 -> \2/g' cmd.txt`
- Take advantage of *pipelines*
- A simple strategy could be to isolate letters (section 2.1) and store them in an array (section 2.4)
- Above commands might have to be adjusted to meet the requirements and avoid bugs