Autonomous Systems - Homework 1

Christoph Killing

Oktober 20, 2021

1 Summary of Shell commands

This is for your reference. Feel free to skip this chapter if you are proficient with terminal already.

1.1 Exploring the file system

1.1.1 pwd

Modern file-systems are organized in folders, being able to navigate the file-system is fundamental. Every time we work with the shell we are within one folder, to know where we are we can use the command pwd (Print Working Directory)

```
$ pwd
/home/markusryll/autonomous_system_2020/lab1
```

1.1.2 ls

To list the contents of the current directory (files and/or child directories, etc.) we use ls (LiSt)

```
$ ls
ex0.cpp   ex1.cpp   ex2.cpp   final.cpp
```

1.1.3 File permissions and ownership

The concept of permissions and ownership is crucial in a unix system. Every to file and directory is assigned 3 types of owner:

- *User*: is the owner of the file, by default, the person who created a file
- *Group*: user-group can contain multiple users, all users belonging to a group will have the same access permissions to the file
- *Other*: Any other user who has access to a file

At the same time to every file and directory is assigned a type of permission

- Read
- Write
- Execute

We get all this information using ls -l, for example:

```
$ ls -l
total 1112
-rw-r--r--  1 markus  staff  557042 Aug 24 21:57 dante.txt
-rwxr-xr-x  1 markus  staff      40 Aug 23 18:36 hello.sh
-rw-r--r--  1 markus  staff     171 Aug 23 18:28 hello_.tar.gz
-rw-r--r--  1 markus  staff      49 Aug 24 22:55 numbers.txt
```

The permissions are specified by the 1st field, the ownership is specified by the 3rd and 4th fields. For example, the file hello.sh is owned by me (markus) and the group is staff. The permission string is -rwxr-xr-x meaning that:

- The owner can read (r), write (w) and execute (x) the file
- The group can read and execute
- Other can read and execute

1.1.4 cd

To change the current folder we can use cd (Change Directory). For example cd / moves to the file system root or

```
cd /home
```

To move to the parent of the current folder we use cd .., it can also be concatenated like cd ../.. to move two (or more) levels up. To move back to your home folder we use cd   (or simply cd).

1.1.5 find

Imagine you have a folder containing many files and you want to locate a file called findme.txt. To accomplish it you can use

```
find . -name "findme.txt"
```

Let's analyze the command. The . represent the current folder, so we are saying to find to look in the current folder recursively (you can change it with relative or absolute paths) for a file called findme.txt. Find is a powerful tool, you can have complex expression to match files, have a look at find –help.

1.2 Edit Filesystem

1.2.1 mkdir

mkdir (make directory) is used to create new, empty directories: let's create a new dir named newdir

```
$ mkdir newdir
$ ls
newdir
$ cd newdir
```

1.2.2 touch

touch was created to modify file time-stamps, but it can also be used to quickly create an empty file. You can easily create a newfile.txt with

```
$ touch newfile.txt
$ ls
newfile.txt
```

1.2.3 rm

You can remove any file with rm – be careful, this is non-recoverable! I suggest to add the flag -i to prompt a confirmation message

```
rm -i newfile.txt
rm: remove regular empty file 'newfile.txt'? y
```

You can also remove directories with rm, the only catch is that it returns an error when the folder is not-empty. The common practice, but pretty prone to non-recoverable errors, is to run rm -rf folder-name. The command will remove the folder with all its content (r - recursive) forcing the operation (f - force). This operation will not ask for confirmation. You can of course add the flag i (i.e. rm -rfi foldername) but will ask confirmation for every file, this is pretty annoying if the folder contains many files.

1.2.4 cp

Copying file is as simple as running cp (CoPy). If we want to duplicate the file numbers.txt we can run

```
$ cp numbers.txt numbers_copy.txt
$ls
numbers.txt  numbers_copy.txt
```

1.2.5 mv

If we want to rename numbers_copy.txt to new_numbers.txt we can run

```
$ mv numbers_copy.txt new_numbers.txt
$ ls
new_numbers.txt  numbers.txt
```

With the same command we can also move the file to another location, for example if we want to move numbers.txt to a newly create folder dataset we execute

```
$ mkdir dataset
$ mv numbers.txt dataset/numbers.txt
$ ls dataset
numbers.txt
```

1.3 Viewing and Editing Files

subsubsectioncat cat concatenates a list of files and sends them to the standard output stream and is often used to quickly view the content of a file. For example we can inspect the content of the file numbers.txt.

```
$ cat numbers.txt
One
Two
Three
Four
Five
Six
Seven
Eight
Nine
Ten
```

1.3.1 nano and vim

nano is a minimalistic command-line text editor. It's a great editor for beginners. More demanding user prefer vim. It's a powerful and highly customizable text editor (I love it!). I strongly suggest to learn how to use vim, one of the best way to learn vim is to simply run vimtutor in your terminal but if you prefer games try Vim Adventures!

1.4 Pipe

The Pipe is a command in Linux that lets you use two or more commands such that output of one command serves as input to the next. In short, the output of each process directly as input to the next one like a pipeline. The symbol — denotes a pipe.

For example, consider the following file:

```
$ cat numbers.txt
One
Two
Three
Four
Five
Six
Seven
Eight
Nine
Ten
```

We can sort the lines piping cat with sort

```
$ cat numbers.txt | sort
Eight
Five
Four
Nine
One
Seven
Six
Ten
Three
Two
```

1.5 Output redirect

We redirect the output of a command to a file. This is useful when we want to save the output of a program without writing specific code.

The common commands that we use and their results are

```
command > output.txt
```

The standard output stream will be redirected to the file only, it will not be visible in the terminal. If the file already exists, it gets overwritten.

```
command &> output.txt
```

Both the standard output and standard error stream will be redirected to the file only, nothing will be visible in the terminal. If the file already exists, it gets overwritten.

```
command | tee output.txt
```

The standard output stream will be copied to the file, it will still be visible in the terminal. If the file already exists, it gets overwritten.

```
command |& tee output.txt
```

Both the standard output and standard error streams will be copied to the file while still being visible in the terminal. If the file already exists, it gets overwritten.

2 Exercises

2.1 Exercise 1: Git

1. Clone the course repository from Autonomous Systems GitLab
2. Create a personal repository to work on the homework problems on GitLab. Also clone that to your machine.
3. Create a folder named Lab_1 and copy the content from the repository 'autonomous-systems-2021\Labs\Lab_1' there.
4. Work on Exercise 2, after you are done safe your progress by pushing it to the git.

2.2 Exercise 2: C++

The goal of this exercise is to update your C++ skills and to learn how to compile and run a program under Linux.

We will first need to install build-essentials from the terminal to later compile our code. Please run:

```
sudo apt-get install build-essential
```

In this exercise we will implement the class RandomVector. Inside /yourRepository/lab1 create a folder called RandomVector and copy the content from autonomous-systems-2020/Labs/Lab_1/Code.

The class RandomVector defined in the header file random_vector.h abstract a vector of doubles. You are required to implement the following methods:

- RandomVector(int size, double max_val = 1) (constructor): initialize a vector of doubles of size size with random values between 0 and max_val (default value 1)
- double mean() returns the mean of the values in random vector
- double max() returns the max of the values in random vector
- double min() returns the min of the values in random vector
- void print() prints all the values in the random vector
- void printHistogram(int bins) computes the histogram of the values using bins number of bins between min() and max() and print the histogram itself (see the example below).

To to so complete all the TODOs in the file random_vector.cpp. When you are done compile the application by running

```
g++ -std=c++11 -Wall -pedantic -o random_vector main.cpp random_vector.cpp
```

If you complete correctly the exercise you should see something like

```
$ ./random_vector
0.458724 0.779985 0.212415 0.0667949 0.622538 0.999018 0.489585 0.460587 0.0795612 0.185496 0.629162 0.3
Mean: 0.393617
Min: 0.0667949
Max: 0.999018
Histogram:
***     ***
***     ***
***     ***
***     ***
***     ***
***     ***
***     *** ***
*** *** *** *** ***
```

2.3 Groups

From Lab 3 onwards, the exercises shall be done in groups of five to six students. You now have two weeks to find yourself a group. The deadline to register your group is November 3rd, 2021 at 23:59.

2.3.1 How to register your group

To register your group, one of you has to create a repository on GitLab. In Gitlab you can add members to a repository and give everyone particular rights. Please add all your group-members to the repository as maintainers. The name of the repository shall be autonomous-systems-2021-group-[whatever fancy name you like]. In the README.md file, please make an introductory paragraph containing the full names, immatrikulation number and LRZ abbreviation (i.e. ab123cde) of all team members. You will also need to share your group repository with the following people in Maintainer status:

- Prof. Markus Ryll @14A9B6D
- Christoph Killing @ga54xas
- Batuhan Yumurtaci @ga42say

2.3.2 How to submit your work

You will use the gitlab repository you just created to collaborate with your teammates (its never to early to learn a proper git workflow) and submit your homework. We will come back to the groups you will work in later but please start thinking about whom you want to work with now.