Programming in the Unix Environment
DV1457/DV1578

—

Lab 1: Commands and Shell Scripting

Sai Prashanth Josyula
Blekinge Institute of Technology

August 27, 2022

The objective of this laboratory assignment is for you to practice shell scripting (lecture 2) and gain experience in writing shell scripts that can accomplish a variety of tasks.

The laboratory assignments should be conducted, solved, implemented, and presented in groups of two students!

Preparations

Read through these laboratory instructions carefully and make sure that you understand what you are supposed to do. If something is unclear, please contact the teaching assistants or the main teacher. You are expected to have acquired the following knowledge before the lab sessions:

- You are supposed to have basic knowledge about working in a Unix environment.

You are expected to find and learn concepts required to complete this assignment. Do not forget to consult the relevant sections in the course book if required. *You are not expected to have experience in shell scripting but are expected to gain experience as you try completing the tasks in this assignment.*

Plagiarism and collaboration

You are expected to work in groups of two. Groups larger than two are not accepted. You are also not expected to work alone unless you have a good reason.

You are allowed to discuss problems and solutions with other groups. However, the code that you write must be your own and cannot be copied or downloaded from somewhere else, e.g., fellow students, the internet, etc. You may copy idioms and snippets from various sources as it is, e.g., snippets such as `awk '{print $1}'`. However, you must be able to clearly describe **each** part of your scripts. The submitted solutions should be developed by the group members only.

This assignment has four tasks, each of which is described below.

Task 1. Write a script to find if a given date actually exists on the calendar

In this task, you will write a simple shell script that can determine if the input date is valid or not. The main challenge is to take the leap years into account. You may use the following logic[a]:

1. If the year y is divisible by 4, go to step 2. Otherwise, go to step 5.
2. If the year y is divisible by 100, go to step 3. Otherwise, go to step 4.
3. If the year is divisible by 400, go to step 4. Otherwise, go to step 5.
4. The year y is a leap year.
5. The year y is not a leap year.

Apart from some obvious checks, you will also need to check whether the day value in the input date exceeds the number of days in the input month. If it does, then the input date does not exist on the calendar.

Hint: You should be able to write your script using `case`, `echo`, and `if else`.

[a] `https://docs.microsoft.com/en-us/office/troubleshoot/excel/determine-a-leap-year`

The output of your script should look like this:

```
$ ./datecheck.sh nov 28 1992
EXISTS! Nov 28 1992 is someone's birthday.
$ ./datecheck.sh 11 31 2022
BAD INPUT: Nov does not have 31 days.
$ ./datecheck.sh sep 36 2022
BAD INPUT: Sep does not have 36 days.
$ ./datecheck.sh feb 29 2018
BAD INPUT: Feb 2018 does not have 29 days: not a leap year.
$ ./datecheck.sh Feb 29 2000
EXISTS! Feb 29 2000 is someone's birthday.
```

Task 2. Write a script to convert a given number to bytes, kilobytes, megabytes, and gigabytes

In this task, you will write a simple shell script that can print a given input in bytes (B), kilobytes (KB), megabytes (MB), and gigabytes (GB). You must use the utility `bc` in your shell script to do the calculations. The formulae are as follows:
1 KB = 1024 B
1 MB = 1024 KB
1 GB = 1024 MB

The output of your script should look like this:

```
$ ./convertsize.sh 4096B
Bytes = 4096
Kilobytes = 4.0000
Megabytes = .0039
Gigabytes = 0
$ ./convertsize.sh 4096KB
Bytes = 4194304
Kilobytes = 4096
Megabytes = 4.00
Gigabytes = .0039
```

```
$ ./convertsize.sh 4096MB
Bytes = 4294967296
Kilobytes = 4194304
Megabytes = 4096
Gigabytes = 4.0000
$ ./convertsize.sh 4096GB
Bytes = 4398046511104
Kilobytes = 4294967296
Megabytes = 4194304
Gigabytes = 4096
```

Task 3. Write a script to create a compressed version of a specified directory

In this task, you will take the role of a system administrator who wants to backup specific directories. You will write a simple shell script that can create a compressed tar archive of any directory that is given as input. In your script, you may of course use the `tar` command which is capable of archiving a directory. However, your script must fulfill the following requirements:

1. Your script should accept exactly one argument, and if executed without any argument, it should display a message.
2. Your script should check whether the directory given as the argument exists or not.
3. Your script should ensure that the user is in the parent directory of directory to be compressed.
4. Your script should save the archive file in the parent directory of the compressed directory.
5. Your script should verify whether the user who invoked the script has `write` permission on the current directory. Otherwise, it should print the following message: "cannot write the compressed file to the current directory".
6. Your script should warn users before archiving a large directory. The warning message should be as follows: "Warning: the directory is 512 MB. Proceed? [y/n]"
   Hint: Use the `du` command and `awk` to get the size of the directory to be compressed and compare it with 512 MB.

The output of your script should look like this:

```
$ ls
compdir.sh  testdir
$ $./compdir.sh
No arguments given! Give a directory as input
$ ./compdir.sh testdire
compdir.sh: cannot find directory testdire
$ $./compdir.sh testdir
Directory testdir archived as testdir.tgz
$ ls
compdir.sh  testdir testdir.tgz
$ ./compdir.sh testdir/dir1
compdir.sh: cannot find directory testdir/dir1
$ cd testdir
$ mkdir dir1
$ cd ..
$ ./compdir.sh testdir/dir1
compdir.sh: you must specify a subdirectory
$ ./compdir.sh testdir
```

```
Directory testdir archived as testdir.tgz
```

Task 4. Write a script that can run different commands in parallel

In this task, you will practice how to make use of multiple cores of the underlying machine from a shell script. The task is as follows. Your script should try compressing a given large file with four different tools (`gzip`, `bzip2`, `p7zip`, and `lzop`). Your script shall then retain the compressed file that is the smallest and report the result to the user. The script shall remove all other compressed files and the original file as well. Note that at times the compressed files may all be larger than the original file. In that case, your script should display a relevant message and retain the original file.

Your script should compress the file using the above four tools *in parallel*. This is actually quite simple and can be achieved using a special character.

The output of your script should look like this:

```
$ ls -l largefile.txt
-rw-r--r-- 1 ubuntu ubuntu 944824 Aug 7 18:24 largefile.txt
$ compress.sh largefile.txt
Most compression obtained with gzip. Compressed file is largefile.txt.gz
$ ls -l largefile.txt
ls: cannot access 'largefile.txt': No such file or directory
$ ls -l largefile.txt.gz
-rw-r--r-- 1 ubuntu ubuntu 66264 Aug 7 18:24 largefile.txt.gz
```

Presentation and Grading

The assignment should be presented in person with both group members in attendance on a lab slot. You will be required to show the code, let me run it and be prepared to discuss your work. You are expected to perform thorough tests on your scripts before submission. You should demonstrate a clear understanding of the problem and detail your strategy of how to solve it, including drawbacks and advantages of the devised solution. Your script should be clear, concise and to the point.

Grading: If you complete and submit any two of the four tasks, you will qualify for *Grade D*. If you complete any three of the four tasks, you will qualify for *Grade C*. If you complete all the tasks, you will qualify for *Grade B*.

Note: Even if your implementation fulfills the requirements for the grade you were targeting, you might get a lower grade (lower by one), if your code is of bad quality. For this reason, there are no requirements for Grade E. You can only get an E, if your implementation fulfills the requirements for Grade D and your code is of bad quality. In this way, it is impossible to fail the assignment just because of poor code quality.

Finally, if you do all the tasks and have well-commented, clearly understandable scripts that solve the problem elegantly, you will be awarded ***Grade A***. The programs in this lab assignment will be tested on Ubuntu 22.04 Virtual machine.

———*All the best!*———