Management of Information Systems
2020

About the laboratory

This laboratory guide contains 14 exercises. After each exercise you will find some ideas, some commands that might be handy during the solution. But, of course, any resources available on the Internet may be used.

The next few tutorials may be helpful during preparing for the laboratory:

1. https://www.digitalocean.com/community/tutorials/how-to-set-up-a-firewall-using-iptables-on-ubuntu-14-04
2. https://www.digitalocean.com/community/tutorials/how-to-set-up-apache-virtual-hosts-on-ubuntu-16-04
3. https://www.shellscript.sh

Starting the VM

During the laboratory you will use a virtual machine preinstalled with a Debian 9.0 system. Note that the virtual machine has no internet connection, only a direct connection to the host computer is present. Please do not change this setting. If you have to move files between the host and the virtual system, please use the WinSCP installed on the host computer.

You should login as ***laboruser*** to the virtual machine.
The password of *laboruser* is *laboruser*.

Some exercises might require administrative privileges.
The root password is *irulabor*.

~~ Exercises ~~

1.1 Check the codename of the system.

lsb_release -da

1.2 List all the installed packages. Is the Midnight Commander file manager installed?

dpkg -l

1.3 Check the network interfaces of the virtual machine! What is the IP address of each of them?

ip addr

1.4 According to the policies the *ssh* service should be reachable only from the organization's network. Modify the firewall rules according to the policy!

iptables -L INPUT, iptables -A INPUT …

1.5 According to the policies the machine should not answer the echo requests ('ping'). Modify the firewall rules according to this policy!

1.6 Create a new user called *spongebob*. The password of *spongebob* should be *abracadabra11*.

adduser

1.7 The new user created in the previous exercise should be allowed the sud rights!

sudo

2.1 Install the Apache2 webserver!

apt-get install apache2

2.2 Check that the webserver is listening on the port 80!

lsof -i -P

2.3 Create a new virtual host that answers the name *irulabor.vmware*. The name *irulabor.vmware* should be resolved to 127.0.0.1 by the virtual machine! The webserver should return the pages you can download from the address https://github.com/ng201/iru.

/etc/hosts, a2ensite

2.4 You can check the working configuration on the virtual and on the physical machine, too. The page *http://irulabor.vmware/vedett* can be reached from both machines. Change the settings of the virtual host in a way that the content of the *vedett* directory is available from the virtual machine (i.e., from the IP address 127.0.0.1)!

Require ip 127.0.0.1

3.1 Write a *bash* script that prints the vendor id of the processor!

cat /proc/cpuinfo, cut

3.2 Write a bash script that's input is a file of 5 columns (the number of rows is unspecified). The columns are separated by spaces and the first column contains a positive integer value. The script should i) concatenate

the 4th and 5th columns, ii) multiply the number I the first column by 2 and iii) write out the result on the standard output as a 4-column dataset.

awk

3.3 Write a *bash* script that prints out i) the name of user running the script, ii) the current date in YYYY-mm-dd format and iii) the list of users logged in to the system. Each user should be only once on the list!

who, date +

Scripting tutorial

Some useful programs

In this section we introduce some useful programs that can serve as building blocks to write great scripts.

cat

cat concatenates files and prints the result on the standard output.

head

Head is a program that prints the first so many lines of its input. By default, it will print the first 10 lines, but this behaviour can be overwritten with a command line argument.

tail

Tail is the opposite of head. Tail is a program that prints the last few lines of its input. By default, it will print the last 10 lines.

```
laboruser@iru:~$ tail -3 mydata.txt
Greg pineapples 3
Betty limes 14
laboruser@iru:~$
```

sort

Sort will sort its input. By default, it will sort alphabetically but there are many options available to modify the sorting mechanism. See, for example, the man page of sort for more details.

wc

wc stands for word count and it can print newline, word, and byte counts for each file.

cut

cut is a little program to separate the content of a file into field (columns). The field separator character may be defined with the -d command line option. The -f option allows us to specify which field or fields we would like to see in the output. If we want the first two column of a space separated file, we can run the command as follows:

```
laboruser@iru:~$ cut -f 1,2 -d ' ' mydata.txt
```

uniq

uniq stands for unique and its job is to remove duplicate lines from the data. One limitation however is that those lines must be adjacent.

grep

grep is a program which will search a given set of data and print every line which contains a given pattern. It has many command line options which modify its behaviour, so it's worth checking out its man page:

```
laboruser@iru:~$ grep [command line options] <pattern> [path]
```

In the examples below we will use a sample file as follows:

```
laboruser@iru:~$ cat mydata.txt
Fred apples 20
Susy oranges 5
Robert pears 4
Terry oranges 9
Lisa peaches 7
Susy oranges 12
Mark grapes 39
Anne mangoes 7
Betty limes 14
```

Let's say we wished to identify every line which contained the string oranges:

```
laboruser@iru:~$ grep 'oranges' mydata.txt
Susy oranges 5
Terry oranges 9
Susy oranges 12
```

The basic behaviour of grep is that it will print the entire line for every line which contains a string of characters matching the given pattern. This is important to note, we are not searching for a word but a string of characters.

Also note that we included the pattern within quotes. This is not always required. They are required if your pattern contains characters which have a special meaning on the command line.

Sometimes we are not interested in seeing the matched lines but wish to know how many lines did match.

```
laboruser@iru:~$ grep -c 'orange' mydata.txt
3
```

sed

sed (stream editor) is a Linux/Unix utility that parses and transforms text. The following example shows a typical, and the most common, use of sed, for substitution:

```
laboruser@iru:~$ sed 's/regexp/replacement/g' inFName > outFName
```

Besides substitution, other forms of simple processing are possible, too. For example, the following uses the d command to delete lines that are either blank or only contain spaces:

```
laboruser@iru:~$ sed '/^ *$/d' inFName
```

This example uses some of the following regular expression:

- the caret (^) matches the beginning of the line.
- the dollar sign ($) matches the end of the line.
- the asterisk (*) matches zero or more occurrences of the previous character.

awk

Awk is a good program to use if you need to work with data that is organized into records with fields. It allows you to filter the data and control how it is displayed.

Now let's say we want to print the order but only if the order is above 10. We also wish to reformat the output a little.

```
laboruser@iru:~$ awk '$3 > 10 {print $1 " - " $2 ": " $3}' mydata.txt
Fred - apples: 20
Susy - oranges: 12
Mark - grapes: 39
Betty - limes: 14
```

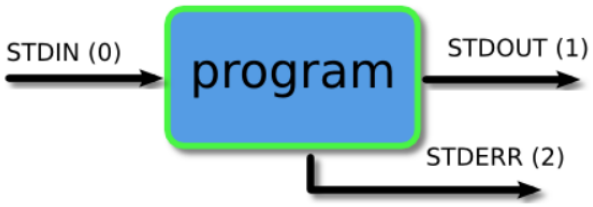
Piping and Redirection

Every program we run on the command line automatically has three data streams connected to it.

STDIN (0) - Standard input (data fed into the program)

STDOUT (1) - Standard output (data printed by the program, defaults to the terminal)

STDERR (2) - Standard error (for error messages, also defaults to the terminal)



Piping and redirection is the means by which we may connect these streams between programs and files to direct data in interesting and useful ways.

Redirecting to a File

Normally, we will get our output on the screen, which is convenient most of the time, but sometimes we may wish to save it into a file to keep as a record, feed into another system, or send to someone else. The greater than operator (>) indicates to the command line that we wish the programs output

(or whatever it sends to STDOUT) to be saved in a file instead of printed to the screen. Let's see an example.

```
laboruser@iru:~$ ls
sponge.txt bob example.png firstfile foo1 video.mpeg
laboruser@iru:~$ ls > myoutput
laboruser@iru:~$ ls
laboruser@iru:~$  sponge.txt  bob  example.png  firstfile  foo1  myoutput
video.mpeg
laboruser@iru:~$ cat myoutput
sponge.txt
bob
example.png
firstfile
foo1
myoutput
video.mpeg
```

Piping

Piping is a mechanism for sending data from one program to another. The operator we use is |. What this operator does is feed the output from the program on the left as input to the program on the right. In the example below we will list only the first 3 files in the directory.

```
laboruser@iru:~$ ls
sponge.txt bob example.png firstfile foo1 myoutput video.mpeg
laboruser@iru:~$ ls | head -3
sponge.txt
bob
example.png
```

We may pipe as many programs together as we like:

```
laboruser@iru:~$ ls | head -3 | tail -1
example.png
```

Bash Scripting

Bash is a Unix/Linux shell and command language. Bash is a command processor that typically runs in a text window where the user types commands that cause actions.

Bash can also read and execute commands from a file, called a shell script. Like all Linux/Unix shells, it supports filename globbing (wildcard matching), piping, here documents, command substitution, variables, and control structures for condition-testing and iteration.

The Shebang

The very first line of a script should tell the system which interpreter should be used on this file. It is important that this is the very first line of the script. The first two characters #! (the shebang) tell the system that directly after it will be a path to the interpreter to be used.

```
#!/bin/bash
```

Comments

A comment is just a note in the script that does not get run, it is merely there for your benefit. Comments are easy to put in, all you need to do is place a hash (#) then anything after that is considered a comment. A comment can be a whole line or at the end of a line.

```
laboruser@iru:~$ cat myscript.sh
#!/bin/bash
# A comment which takes up a whole line
ls # A comment at the end of the line
```

Variables

A variable is a container for a simple piece of data. Variables are easy to set and refer to but they have a specific syntax that must be followed exactly for them to work.

When we set a variable, we specify its name, followed directly by an equal sign (=) followed directly by the value. Thus, no spaces on either side of the = sign.

When we refer to a variable, we must place a dollar sign before the variable name.

```
laboruser@iru:~$ cat variableexample.sh
#!/bin/bash
# A simple demonstration of variables
# Ryan 29/3/2019

name='SpongeBob'
echo Hello $name

laboruser@iru:~$ ./variableexample.sh
Hello SpongeBob
```

Command line arguments

When we run a script, there are several variables that get set automatically for us. Here are some of them:

$0 - The name of the script.

$1 - $9 - Any command line arguments given to the script. $1 is the first argument, $2 the second and so on.

$# - How many command line arguments were given to the script.

$* - All of the command line arguments.

```
laboruser@iru:~$ cat morevariables.sh
#!/bin/bash
# A simple demonstration of variables
# Ryan 29/3/2019
```

```
echo My name is $0 and I have been given $# command line arguments
echo Here they are: $*
echo And the 2nd command line argument is $2

laboruser@iru:~$ ./morevariables.sh bob fred sally
My name is morevariables.sh and I have been given 3 command line arguments
Here they are: bob fred sally
And the 2nd command line argument is fred
```

Back ticks or $()

It is also possible to save the output of a command to a variable and the mechanism we use for that is the backtick (`) or the $() construct:

```
laboruser@iru:~$ cat backticks.sh
#!/bin/bash
# A simple demonstration of using backticks
# Ryan 29/3/2019

lines=`cat $1 | wc -l`
# or you may use:
# lines=$(cat $1 | wc -l)
echo The number of lines in the file $1 is $lines

./backticks.sh testfile.txt
The number of lines in the file testfile.txt is 12
```