Universiteit Antwerpen
Faculteit Toegepaste Ingenieurswetenschappen

Lab of 3-Network Architecture

Ruben Nietvelt, Nabeel Nisar Bhat

2024-2025

Scheduled labs for PR01

Session	Date	Subject	Evaluation	Deadline (23:59)
1	01/10/2024	Introduction to the Linux Operating System: Using the shell & exploring the filesystem	Report	07/10/2024
2	08/10/2024	Working with text files and managing running processes	Report	14/10/2024
3	15/10/2024	Writing shell scripts	Report	22/10/2024
4	23/10/2024	Learning system administration, getting & managing software	Report	28/10/2024
5	29/10/2024	Wireshark introduction	Report	05/11/2024
6	06/11/2024	Protocols in action: TCP and UDP	Report	11/11/2024
7	12/11/2024	Ethernet and ARP	Report	19/11/2024
8	20/11/2024	Setting up a DHCP server	Report	25/11/2024
9	26/11/2024	Setting up a DNS server	Report	03/12/2024
10	04/12/2024	Network Address Translation	Report	09/12/2024
11	10/12/2024	Remote Access & Firewalls (1)		**N/A**
12	18/12/2024	Remote Access & Firewalls (2)	Blackboard test	

Scheduled labs for PR02

Session	Date	Subject	Evaluation	Deadline (23:59)
1	02/10/2024	Introduction to the Linux Operating System: Using the shell & exploring the filesystem	Report	08/10/2024
2	09/10/2024	Working with text files and managing running processes	Report	15/10/2024
3	16/10/2024	Writing shell scripts	Report	22/10/2024
4	23/10/2024	Learning system administration, getting & managing software	Report	29/10/2024
5	30/10/2024	Wireshark introduction	Report	05/11/2024
6	06/11/2024	Protocols in action: TCP and UDP	Report	12/11/2024
7	13/11/2024	Ethernet and ARP	Report	19/11/2024
8	20/11/2024	Setting up a DHCP server	Report	26/11/2024
9	27/11/2024	Setting up a DNS server	Report	03/12/2024
10	04/12/2024	Network Address Translation	Report	10/12/2024
11	11/12/2024	Remote Access & Firewalls (1)		**N/A**
12	18/12/2024	Remote Access & Firewalls (2)	Blackboard test	

Session 3

Writing shell scripts

Overview

1. Create a file
2. Set your interpreter
3. Save, close and reopen to enable syntax highlighting
4. Populate your script's contents
5. Save your script
6. Make your script executable
7. Execute your script!

1. Create a file

Either create a file

- using the touch command, then edit it

```
touch echotest
$ vim echotest
```

or

- immediately use a text editor, which will create one for you

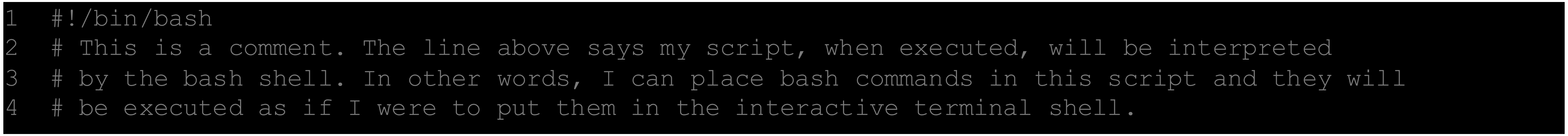
```
$ vim echotest
```

or

```
$ nano echotest
```

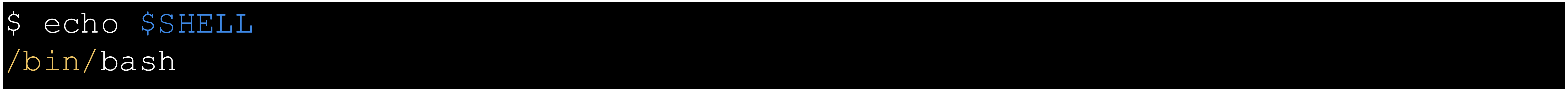
2. Set the interpreter

A shebang (#!) sequence on the first line sets the interpreter



```
#!/bin/bash
# This is a comment. The line above says my script, when executed, will be interpreted
# by the bash shell. In other words, I can place bash commands in this script and they will
# be executed as if I were to put them in the interactive terminal shell.
```

What is the default interpreter in your terminal?



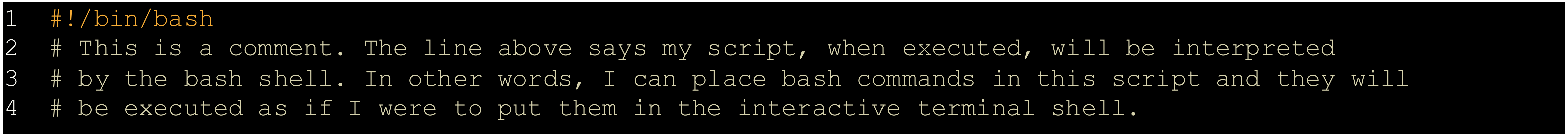
```
$ echo $SHELL
/bin/bash
```

3. Reopen the file to enable syntax highlighting

To save and reload in vim:

1. **:w** (write)
2. **:e** (edit; reloads)

And now the syntax highlighting is enabled based on the interpreter



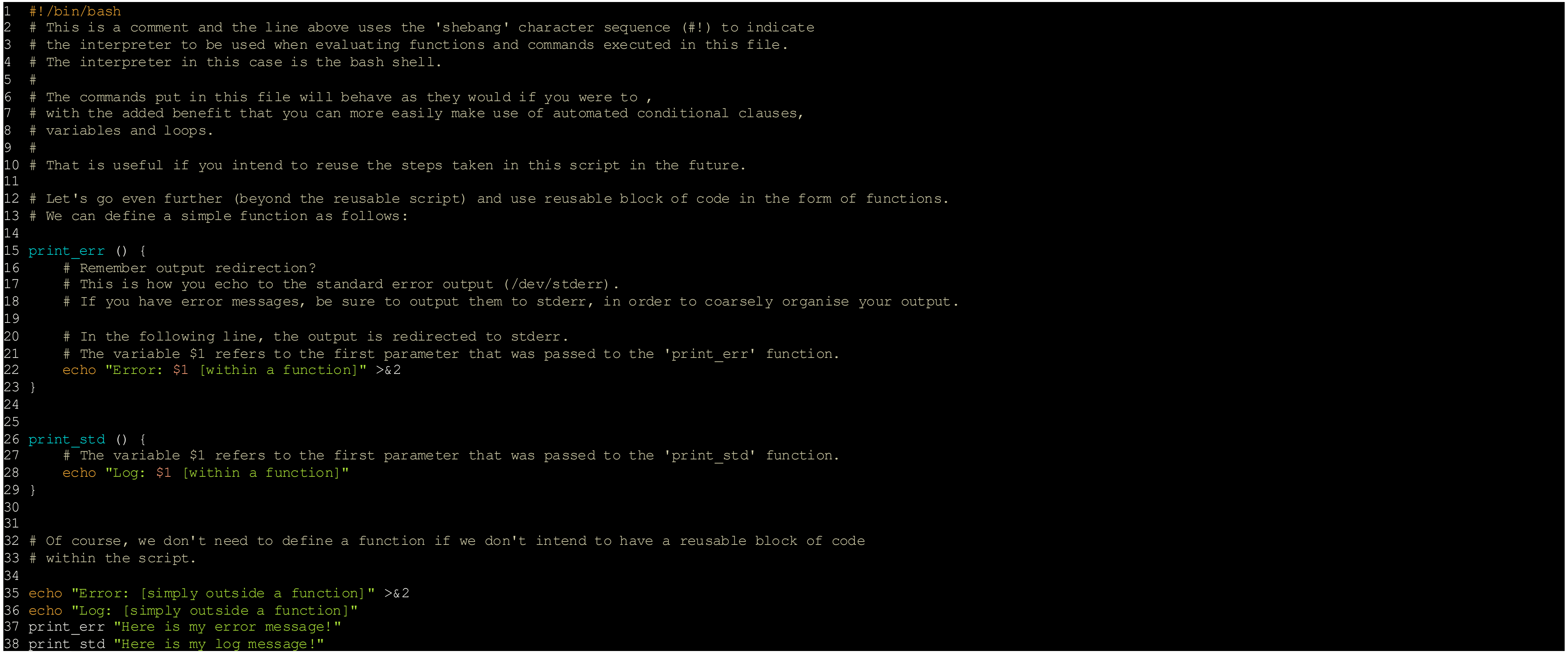
```
#!/bin/bash
# This is a comment. The line above says my script, when executed, will be interpreted
# by the bash shell. In other words, I can place bash commands in this script and they will
# be executed as if I were to put them in the interactive terminal shell.
```

4. Populate your script's content

- shell variables ($)
- escaping shell characters (\)
- positional parameters ($0, $1,…)
- programming constructs
  - *if..then* statements
  - case construct
  - *for..do*, *while..do* and *until..do* loops

```
#!/bin/bash

# Shell Variables and Escaping Characters
NAME="My name is \"Ruben\"."
echo $NAME

# Positional Parameters
echo "Script name: $0, First param: $1, Second param: $2"

# If..then statement
if [ "$1" = "hello" ]; then
          echo "You said hello!"
fi

# Case Construct
case "$2" in
          start) echo "Starting...";;
          stop) echo "Stopping...";;
          *) echo "Unknown action: $2";;
esac

# For loop
for i in 1 2 3; do echo "For loop: $i"; done

# While loop
count=1
while [ $count -le 2 ]; do
          echo "While loop: $count"
          ((count++))
done

# Until loop
counter=3
until [ $counter -le 1 ]; do
          echo "Until loop: $counter"
          ((counter--))
done
```

4. Populate your script's content



```
#!/bin/bash
# This is a comment and the line above uses the 'shebang' character sequence (#!) to indicate
# the interpreter to be used when evaluating functions and commands executed in this file.
# The interpreter in this case is the bash shell.
#
# The commands put in this file will behave as they would if you were to ,
# with the added benefit that you can more easily make use of automated conditional clauses,
# variables and loops.
#
# That is useful if you intend to reuse the steps taken in this script in the future.

# Let's go even further (beyond the reusable script) and use reusable block of code in the form of functions.
# We can define a simple function as follows:

print_err () {
    # Remember output redirection?
    # This is how you echo to the standard error output (/dev/stderr).
    # If you have error messages, be sure to output them to stderr, in order to coarsely organise your output.

    # In the following line, the output is redirected to stderr.
    # The variable $1 refers to the first parameter that was passed to the 'print_err' function.
    echo "Error: $1 [within a function]" >&2
}


print_std () {
    # The variable $1 refers to the first parameter that was passed to the 'print_std' function.
    echo "Log: $1 [within a function]"
}


# Of course, we don't need to define a function if we don't intend to have a reusable block of code
# within the script.

echo "Error: [simply outside a function]" >&2
echo "Log: [simply outside a function]"
print_err "Here is my error message!"
print_std "Here is my log message!"
```

Try redirecting stderr to /dev/null. What is the output?

5. Save your script!

6. Make your script executable

What are the default permissions (P) for folders and files?

And so:

$$P_{default} = P_{default} - umask$$

Where

$$P_{base,folder} = 777,$$

$$P_{base,,file} = 666,$$

And

$$unmask = \text{\$umask}$$

```
$ umask
0002
```

And do:

$$P_{default,folder} = 775$$

and

$$P_{default,file} = 664$$

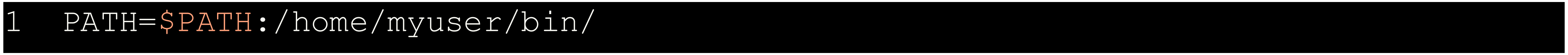
So in other words: by default, the files are not executable, make it executable.

7. Execute your script!

Now that you have made your script executable, how do you run it?

Well, in your command line, simply refer to it:
$ <path>/script
$ script however will not work, unless <path> is in the **$PATH** variable.

Let's create a bin folder in your **$HOME** directory, move our script there and then add this directory to our **$PATH** variable.



```
PATH=$PATH:/home/myuser/bin/
```

Now, **$ script** will execute!

Exercises

Exercises

Solve the following problems using bash scripts. Present your results in a clear manner, but more importantly, present your methods. *Make sure that your script is included in text format and that it can be copy-pasted to validate its workings.*

1. Write a script that takes in **1 argument**, and make sure that it is a positive integer. The script then continues to count down to 0 from that number.

2. Create a folder called "myFiles" with a number of files in it, which you can name yourself.
   Write a script that takes in **1 argument** and checks whether that argument is a **folder**. If so, the script continues to recursively adapt the permissions of the files within to make them only readable and writable by the user, but void of any other permissions.

3. Users have their own user ID, which you can find in the /etc/passwd file. Write a script that takes in **2 arguments**, representing the lower and upper bound of the user ID range to filter. The script then prints the users with user IDs within that range.