Professional Skills
Agile Fundamentals
Jira
Git
Databases Introduction
Java Beginner
Maven
Testing (Foundation)
Java Intermediate
HTML
CSS
Javascript
Spring Boot
Selenium
Sonarqube
Advanced Testing (Theory)
Cucumber
MongoDB
Express
NodeJS
React
Express-Testing
Networking
Security
Cloud Fundamentals
AWS Foundations
AWS Intermediate
Linux
○ Linux Introduction

Scripting in Linux

Contents

Overview

Simply put a script is a file that we can execute.
The file would contain text in the form of commands and execute the commands one after another.

We need to remember to make the command **executable**.

Tutorial

We create a shell script file with the file extension `.sh`

Execute the following command to create a script file called `example.sh`

```
touch example.sh
```

You can use `#` for comments.

```
whoami
# Everything after the hash will be ignored
ls
```

We need to specify how we want the file to be executed, we do this at the top of the file with this line to run the file as a bash script. This line is also known as `shebang`.

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

Then we add the commands we want the script to execute below the shebang line. In the example below the first command `whoami` will print out the username of the current session, and the second command will list all the files within the directory.

```
#! /bin/bash

whoami
ls
```

It is possible to use the `shebang` to specify the execution via a higher level than bash, for instance using `node` or `python` to take advantage of the unique features of these execution environments (sometimes a one line python/node script can do a lot more than any amount of bash scripting, or perhaps you are just more familiar/comfortable manipulating strings using a programming language rather than piping bash processes together).

We then save and exit the file. For the script to be executable, you need to add the executable permission, the command below is specifically for this purpose.

```
chmod +x example.sh
```

Then we can run the file with this command.

```
./example.sh
```

Or we can add the directory that the file resides into the PATH, to make it executable from anywhere.

```
export PATH=$(pwd):$PATH
```

Scripting Logic

If Statements

We can use if statements in our bash scripts. They consist of a statement that if true the code block underneath will be executed.
If not the script will skip the code block.
The code block is initiated with `then` and ends with `fi`.

```
if [ 101 -gt 100 ]
then
    echo "101 is greater than 100"
fi
```

This if statement will always echo "101 is greater than 100" because the conditional `[101 -gt 100]` is true.

These are your different comparisons.

```
-ne #Not equal to
-eq #Equal to
-gt #Greater than
-ge #Greater than or equal to
-lt # Less than
-le #Less than or equal to
```

Variables

You can assign variables in a bash script using `=`.

```
user=johndoe
echo ${user}
```

This will return "johndoe"

User Input

To accept user input as part of a script, we use the `read` command.

```
echo "Please enter your name and press enter"
read inputname

echo "Hello ${inputname}"
```

Then the user's input will be assigned as the variable `inputname`

Loops

There are 2 different kinds of loops in bash scripts. For and while loops.

A for loop will increment through the variables given.

```
echo "Enter 3 names separated with spaces"
read inputnames
personnumber=1

for i in $inputnames
do
    echo "Person number ${personnumber}: ${i}"
    ((personnumber++))
done
```

A while loop will continue to loop the code block until the conditional is no longer true.

```
echo "Enter a number"
read inputnumber
looper=1

while [ ${looper} -lt ${inputnumber} ]
do
    echo "${looper} is smaller then ${inputnumber}"
    ((looper++))
done
```

Switch/Case Statements

Switch statements are useful when you need a large amount of if statements. It also allows you to have a default clause that will occur if none of the other conditionals are true. A common use for switch/case statements is at the beginning of a shell script to capture and validate any arguments that have been passed into the script and providing feedback on requirements.

```
echo "Enter your grade (A-F):"
read grade

case ${grade} in
    "A") echo "Brilliant";;
    "B") echo "Well done";;
    "C") echo "Not bad";;
    "D") echo "Could be better";;
    "E") echo "Not Great";;
    "F") echo "Bad";;
    *) echo "Don't recognise this grade";;
esac
```

Arrays

Arrays are used in bash to contain several elements.
Each element is indexed from 0 onwards.
We create an array with the declare command.

```
declare -a names=("Alice" "Bob" "Luke" "Dylan")
counter=1

for i in "${names[@]}"
do
    echo "Person ${counter}: ${i}"
    ((counter++))
done
```

Exercises

Exercise 1
Write a script that accepts a username as an input and returns if that username is logged in or not.

▶ Answer

Exercise 2
Write a script that does the following:

```
    - Creates a new directory inside your home directory
    - Creates 2 .txt files
    - Moves one of the files outside the directory
    - Renames that file to test1.txt
    - Renames the second file to test2.sh
    - Make test2.sh a script that prints the working directory
    - Execute that script
```

Make this script executable from everywhere.

▶ Answer

Exercise 3

Write a script that takes a URL as user input and then returns whether that website exists.

▶ Answer

Exercise 4

Write a script that takes a file as user input and returns whether the file exists and if so, the full path to that file.

▶ Answer

Exercise 5

Write a script that does the following:

```
- Takes a filename and title as user input
- Creates (if it doesn't exist) a directory named the current month and year
- Creates a markdown file inside that directory with the user's filename and the
day of the month on the end, for example, "hello-17.md".
- While also entering the title and date at the top of the markdown file. So
that when you go to edit the file, the first line says the title and the full
date.
- Stretch Goal: Try and get the script to take you to that directory and stay
there.
```

▶ Answer