Computer Systems and -architecture

Scripting

1 Ba INF 2019-2020

Stephen Pauwels
stephen.pauwels@uantwerpen.be

Deze les is bedoeld om zelfstandig te leren hoe je scripts kan schrijven en uitvoeren op UNIX systemen. Een script is een tekstfile waarin we verschillende commando's kunnen groeperen. Door commando's in een script te zetten zorgen we ervoor dat we eenvoudig dezelfde commando's opnieuw kunnen uitvoeren. Verder kunnen we in een script ook typische programmeer constructies gebruiken zoals variabelen, condities, loops, ...

Script files slaan we typisch op als een `.sh` file. Maak een nieuwe file `test.sh` aan. Vervolgens voegen we volgende lijn toe aan het script:

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

- `#!` geeft aan dat de file een script is
- `/bin/bash` is de shell die we willen gebruiken om ons script uit te voeren
- Als een script uitgevoerd wordt, dan wordt het programma achter `#!` uitegevoerd en wordt de naam van het script hieraan meegegeven als argument
- Aangezien de file begint met `#` wordt deze lijn genegeerd door de shell

Vooraleer we dit script kunnen uitvoeren moeten we ervoor zorgen dat de file ook *uitvoerbaar* wordt. Dit kunnen we doen met het volgende commando:

```
chmod +x test.sh
```

Vervolgens kunnen we het script uitvoeren:

```
./test.sh
```

We krijgen geen output te zien, dit komt omdat er zich geen uit te voeren commando's in het script bevinden. Voeg nu de volgende lijn toe aan het script:

```
3 echo Hello, World!
```

voeren we nu het script uit, dan krijgen we `Hello, World!` als output in de terminal te zien.

Door gebruik te maken van `#` kunnen we commentaar toevoegen aan onze scripts. Alles achter `#` tot het einde van de lijn wordt gezien als commentaar. We krijgen dan bijvoorbeeld tot nu toe volgend script:

```
#! /bin/bash

echo Hello, World!        # Prints Hello, World! to the output
```

1 Variabelen

Als we scripts schrijven kunnen we, net als in Python, gebruik maken van variabelen om informatie in op te slaan. De afspraak is om variabelen in hoofdletters te schrijven, zodat deze goed opvallen in ons script. Willen we nu een variabele gebruiken in het vorige scriptje om de string in op te slaan dan kunnen we dit als volgt doen:

```
OUTPUT_STRING="Hello, World!"
```

We kunnen ook de output van een commando opslaan in een variabele. Dit doen we door gebruik te maken van `$(...)`. Willen we bijvoorbeeld de output van het commando `ls` opslaan, dan kunnen we dit als volgt doen:

```
OUTPUT_LS=$(ls)
```

Het is belangrijk op te merken dat er rond de assignment `=` geen spaties mogen voorkomen. Als we nu de *waarde* van een variabele willen gebruiken moeten we dit in Bash scripting expliciet aangeven door gebruik te maken van `$`. Als we de string die we hebben opgeslagen willen uitprinten dan kunnen we dit als volgt doen:

```
#! /bin/bash

OUTPUT_STRING="Hello, World!"
OUTPUT_LS=$(ls)
echo $OUTPUT_STRING
```

Het teken `$` staat voor *de waarde van*, als we terugkijken naar hoe we de output van `ls` konden opslaan in een variabele, zien we dat we hier ook gebruik maken van `$` om de waarde (output) van het commando te krijgen. We kunnen `echo` ook gebruiken om de output van `ls` weer te geven.

```
#! /bin/bash

OUTPUT_STRING="Hello, World!"
OUTPUT_LS=$(ls)
echo $OUTPUT_STRING
echo $OUTPUT_LS
```

Merk op dat deze output niet zo mooi weergegeven wordt als wanneer we het commando zelf zouden uitvoeren. Dit komt omdat in de variabele de lijst met alle files die zich in de huidige map bevinden worden opgeslagen. Zodat je deze later bijvoorbeeld in een loop kan gebruiken. Je kan variabelen in een string gebruiken, hiervoor zet je de variabele naam best tussen `${...}`

```
#! /bin/bash

OUTPUT_STRING="Hello, World!"
OUTPUT_LS=$(ls)
echo $OUTPUT_STRING
echo "De inhoud van de huidige map is: ${OUTPUT_LS}."
```

Het is ook mogelijk input te vragen aan de gebruiker door gebruik van het keyword `read`. Stel we willen een getal opvragen aan de gebruiker, dan kunnen we dit als volgt doen:

```
#! /bin/bash

echo "Gelieve een getal in te geven: "
read GETAL
echo "Je hebt ${GETAL} ingegeven!"
```

In Bash scripting hebben we ook enkele variabelen met een speciale betekenis:

- `$@`: de lijst van alle positionele parameters
- `$#`: het aantal positionele parameters
- `$1, ..., $9`: de negen eerste positionele parameters
- `$?`: de exit status van het laatst uitgevoerde commando
- `$!`: de PID van het laatst gestarte process in het script
- `$RANDOM`: een random positieve integer

Opdracht 1

Schrijf een script dat je naar je naam vraagt en vervolgens "Hello *naam*" uitprint.

Opdracht 2

Schrijf nu een script dat als extra parameter je naam meekrijgt en vervolgens "Hello *naam*" uitprint.

2 Conditions

Condities worden tussen `[ ... ]` gezet, hierbij is het van belang dat er voor en na de haken wel een spatie gezet wordt. We hebben volgende mogelijke condities die we kunnen gebruiken:

- Files
  - `-e`: file bestaat
  - `-d`: is een directory
  - `-f`: is een reguliere file
  - `-r`: is readible
  - `-w`: is writable

- Strings
  - `-n`: lengte van de string is niet nul
  - `-z`: lengte van de string is gelijk aan nul
  - `s1 = s2`: s1 en s2 zijn identiek
  - `s1 != s2`: s1 en s2 zijn niet identiek
- Getallen
  - `i1 -eq i2`: i1 en i2 zijn gelijk
  - `i1 -ne i2`: i1 en i2 zijn niet gelijk
  - `i1 -gt i2`: i1 is groter dan i2
  - `i1 -ge i2`: i1 is groter of gelijk aan i2
  - `i1 -lt i2`: i1 is kleiner dan i2
  - `i1 -le i2`: i1 is kleiner of gelijk aan i2
- En, of, niet
  - `!`: niet operator
  - `&&`: en operator
  - `||`: of operator

We kunnen deze condities gaan gebruiken in combinatie met If-expressies. Maak een nieuw script aan met de naam `condition.sh`. Stel we willen nakijken of `/home` een directory is, dan kunnen we volgende code gebruiken:

```
#! /bin/bash

if [ -d /home ]
then
        echo "Directory"
fi
```

`if` *`condition`* `...` `then` `...` `fi` is de minimale structuur die we moeten gebruiken om bepaalde lijnen uit te voeren in het geval een bepaalde conditie `true` is. Met `elif` kunnen we extra alternatieve condities toevoegen. Willen we een aparte output genereren wanneer `/home` een file is in bovenstaand script, dan kunnen we het script als volgt uitbreiden:

```
#! /bin/bash

if [ -d /home ]
then
        echo "Directory"
elif [ -f /home ]
then
        echo "File"
fi
```

We kunnen ook een else-clause toevoegen, die uitgevoerd wordt als alle andere condities falen.

```
#! /bin/bash

if [ -d /home ]
then
        echo "Directory"
elif [ -f /home ]
then
        echo "File"
else
        echo "Unknown"
fi
```

We kunnen ook commando's gebruiken als condities, als de exit status van het commando gelijk is aan 0 dan wordt de `then`-clause uitgevoerd.

Opdracht 3

Schrijf een script dat je voornaam en geboortedatum opvraagt. Vervolgens print dit één van twee dingen uit: als het vandaag je verjaardag is, dan print het "Happy Birthday *firstname*" uit, in de andere gevallen print het "Hello *firstname*" uit.

Opdracht 4

Schrijf een script dat een filename inleest en nakijkt of dit een reguliere file of een directory is. Als het een reguliere file is en de file is leesbaar dan geef je de inhoud van deze file weer op het scherm. Als het een directory is, en de directory is leesbaar dan geef je de inhoud van deze directory weer. In alle andere gevallen geef je een error terug.

Een andere conditionele constructie die we in Bash scripting hebben is de Case expressie. Hierbij gaan we bepalen welke code uitgevoerd moet worden aan de hand van aan welk patroon een woord voldoet. Maak een nieuw script `case.sh` aan en bekijk onderstaand voorbeeld in verband met Engelse telwoorden:

```
#! /bin/bash

read NUMBER
case $NUMBER
in
        11|12|13)
                echo ${NUMBER}th
        ;;
        *1)
                echo ${NUMBER}st
        ;;
        *)
                echo ${NUMBER}th
        ;;
esac
```

door gebruik te maken van `|` kunnen we aangeven dat meerdere cases toegelaten zijn. De case `*` staat voor 'al de rest'. Elk blok van uit te voeren commando's beëindigen we met `;;`. Indien er meerdere cases matchen met het opgegeven woord, dan wordt telkens de eerste uitgevoerd.

3 Loops

We hebben ook de mogelijkheid om te itereren over een lijst. Een lijst in Bash scripting kan een lijst van literals `a b c` zijn, een patroon `*.jpeg` (alle `.jpeg` bestanden) of de output van een commando. Het script gaat dan de body uitvoeren voor elk element in de lijst waarbij de loop variabele gelijk is aan het huidige woord. Maak een nieuw script `for.sh` aan en kijk naar volgende code:

```
#! /bin/bash

for LOOP_VAR in a b c
do
        echo $LOOP_VAR
done
```

We zien dat we drie keer de loop uitvoeren en telkens een ander element uit `a b c` uitprinten. Willen we nu itereren over de output van een commando dan kunnen we dit als volgt doen:

```
#! /bin/bash

for LOOP_VAR in $(ls -a)
do
        echo $LOOP_VAR
done
```

Herinner dat `$(...)` moeten gebruiken om de waarde van de output van het commando te verkrijgen, anders gaat het script dit gewoon interpreteren als een lijst van strings.

Opdracht 5

Schrijf een script dat de inhoud van elke file in je huidige directory weergeeft.

Verder hebben we ook conditionele loops. Met `while` kunnen we commando's uitvoeren zolang een conditie true is (of de exit status gelijk is aan 0). Verder hebben we ook `until`, hierbij worden commando's uitgevoerd zolang de contitie false is (of de exit status is verschillend van 0).

```
#! /bin/bash

while condition
do
        commands
done
```

```
#! /bin/bash

until condition
do
        commands
done
```

4 Arithmetic

Bash scripting laat ook toe om wiskundige bewerkingen uit te voeren, deze worden tussen `((...))` gezet. Het is enkel mogelijk om operaties te doen met integers. De exit status van een wiskunde expressie is 0 als het resultaat van de expressie niet gelijk is aan 0 en 1 als het resultaat van de expressie gelijk is aan 0. Ook moeten we hier weer `$` gebruiken als we het resultaat van de expressie willen (net als bij variabelen en commando's). Maak een script `arithmetic.sh` en bestudeer volgende code:

```
#! /bin/bash

A=$RANDOM
B=$RANDOM
C=$A
D=$B

while ((D != 0))
do
        TEMP=$D
        D=$((C % D))
        C=$TEMP
done

echo "The GCD of $A and $B is $C"
```

Opdracht 6

Schrijf een script dat twee nummer inleest en het grootste van de twee uitprint.

Opdracht 7

Schrijf een script dat het spel *nim* implementeerd. Bij het begin van het spel zijn er een aantal lucifers (door de spelers te bepalen). Per beurt neemt één van de spelers 1,2 of 3 lucifers weg. De persoon die de laatste lucifer wegneemt wint het spel.