Computer Systems and Architecture

UNIX Scripting

Stephen Pauwels

Academic Year 2019-2020

Universiteit Antwerpen

Outline

- Basics
- Conditionals
- Loops
- Advanced
- Exercises

Shell Scripts

- Grouping commands into a single file
  - Reusability
- Possible to use programming constructs
  - Variables
  - Conditionals
  - Loops
  - …
- No compilation required

How to create a Shell Script

1. Save script as a (.sh) file
2. Add the line '`#!/bin/bash`' to the beginning of the scipt
   - '`#!`' indicates that the file is a script
   - '`/bin/bash`' is the shell that is used to execute the script
   - When the scipt is executed, the program after the '`#!`' is executed and the name of the script is passed to it
   - Since the line starts with a '#' it is ignored by the shell
3. Make the script executable using '`chmod +x`'
4. Execute the script by calling it
   - Put '`./`' in front of the name in order to avoid confusion with commands

Comments

- Comments are placed behind a # and last until the end of the line
- There are no multiline comments
- The `#!` line is a comment

Variables - Basic

- Assigning a variable
  - `VARIABLE=value`
  - `VARIABLE=$(command -options arguments)`
  - No spaces before and after the '='!
- Using the value of variables
  - Place a '$' before the name
  - If the variable name is followed by text -> place the name between braces
    - E.g.: `echo "Today is the ${DAY}th day of the week"`
- Getting keyboard input
  - `read VARIABLE`
- Exporting variables
  - To make them accessible from other programs
  - Place '`export`' before the name of the variable
  - E.g.: `export PATH='/bin:/usr/bin'`

Variables - Specials

`$@`	Expands to the list of positional parameters separated by commas
`$#`	The number of positional parameters
`$0`	The name of the script
`$1, …, $9`	The nine first positional parameters
`$?`	The exit status of the last executed command
`$!`	The PID of the last process that was started in the script
`$RANDOM`	A positive random integer

Example

script.sh

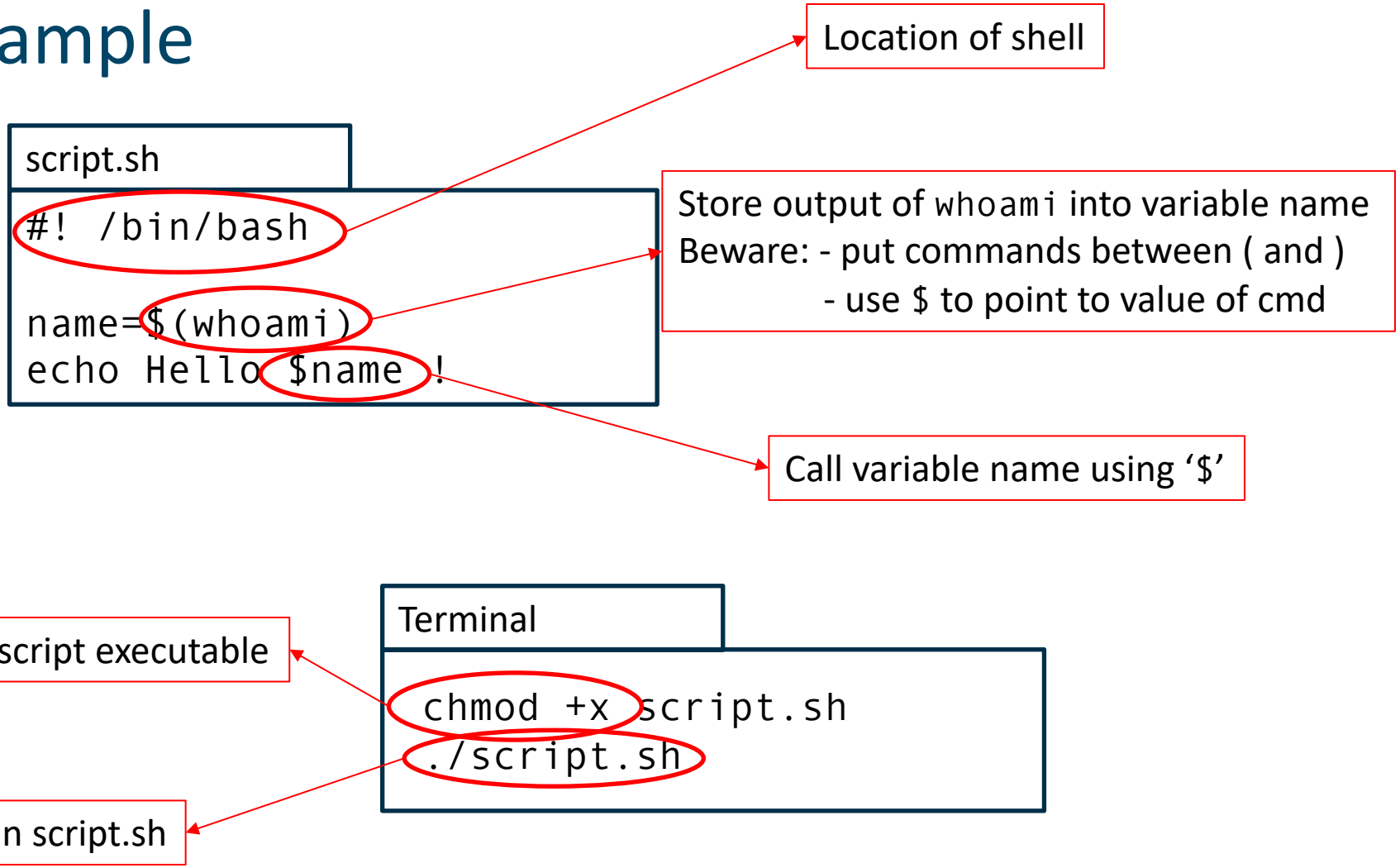
```
#! /bin/bash

name=$(whoami)
echo Hello $name !
```

Terminal

```
chmod +x script.sh
./script.sh
```

Example


Location of shell
script.sh
#! /bin/bash
name=$(whoami)
echo Hello $name !
Store output of whoami into variable name
Beware: - put commands between (and)
- use $ to point to value of cmd
Call variable name using '$'
Terminal
chmod +x script.sh
./script.sh
Make script executable
Run script.sh

Conditions - Basic

- Between `[ … ]`
- Spaces before and after `[ ]`
- Examples
  - `[ -d dir ]` returns `true` if `dir` is a directory
  - `[ $var -eq 2 ]` returns `true` if `$var` equals 2
  - `[ $var -eq 1 ] || [ $var -eq 2 ]` returns `true` if `$var` equals 1 or 2

Conditions

- Files

`-e`	File exists
`-d`	Is a directory
`-f`	Is a regular file
`-r`	Is readible
`-w`	Is Writeable

- Strings

`-n`	Length of string is nonzero
`-z`	Length of string is zero
`s1 = s2`	s1 and s2 are identical
`s1 != s2`	s1 and s2 are not identical

Conditions

- Numbers

`i1 -eq i2`	i1 and i2 are equal
`i1 -ne i2`	i1 and i2 are not equal
`i1 -gt i2`	i1 is greater than i2
`i1 -ge i2`	i1 is greater than or equal to i2
`i1 -lt i2`	i1 is less than i2
`i1 -le i2`	i1 is less than or equal to i2

- And, or, not

!	NOT operator
&&	AND operator
||	OR operator

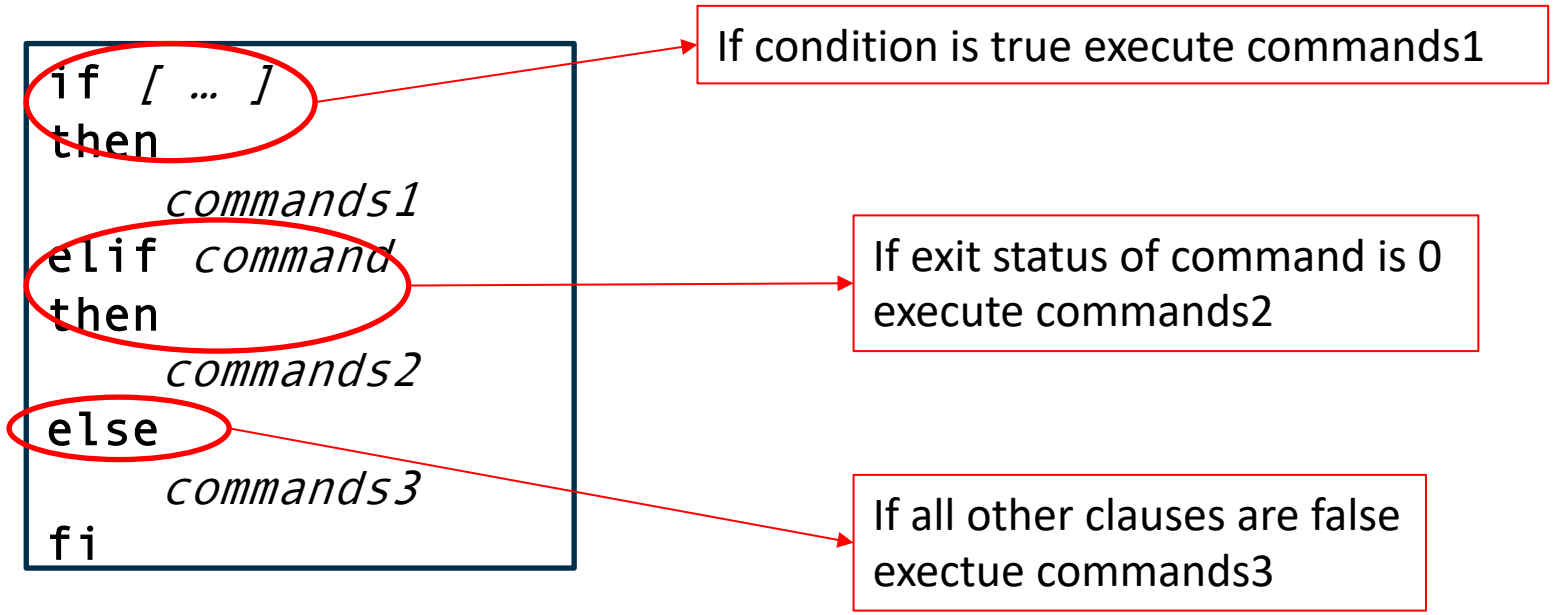
If statements

- Zero or more elif clauses are possible
- The else clause is optional
- The if body is executed if the exit status of the condition is 0

```
if [ … ]
then
    commands1
elif command
then
    commands2
else
    commands3
fi
```

If statements

- Zero or more elif clauses are possible
- The else clause is optional
- The if body is executed if the exit status of the condition is 0



Case statements

- Executes code based on which pattern matches a word
- Multiple cases can be specified per block by separating them using '|'
- Each block has to be terminated by a ';;'
- Use '*' to match 'the rest'
- If multiple cases match, the first one is executed

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

For loops

- The list can be
  - A literal list: a b c
  - A glob pattern: *.jpeg
  - The output of a command: `ls –a`
- The body is executed for each element in the list
- The loop variable is set to the value of the current word

```
for VARIABLE in list
do
    echo $VARIABLE
done
```

While and Until loops

- The condition is evaluated on each iteration
- While loops are executed as long as the exit status of the condition is zero
- Until loops are executed as long as the exit status of the condition is not zero

```
while condition
do
    commands
done
```

```
until condition
do
    commands
done
```

Break and continue

- `Break` casues the loop to be exited immediately
- `Continue` causes a loop to continue with the next iteration
- An integer parameter can be specified to continue or break from the nth enclosing loop
  - ‘`break 2`’ will break from the second enclosing loop
  - ‘`continue 1`’ is the same as ‘continue’

Arithmetic

- Arithmetic can be performed between `((` and `))`
- Only operations on integers are possible
- The exit status is 0 when the result of the expression is not zero and 1 if the result of the expression is zero
- An expression between `$((` and `))` expands to the result of the expression
- For more advanced calculations `bc` can be used

Arithmetic - Example

```
#! /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”
```

Functions

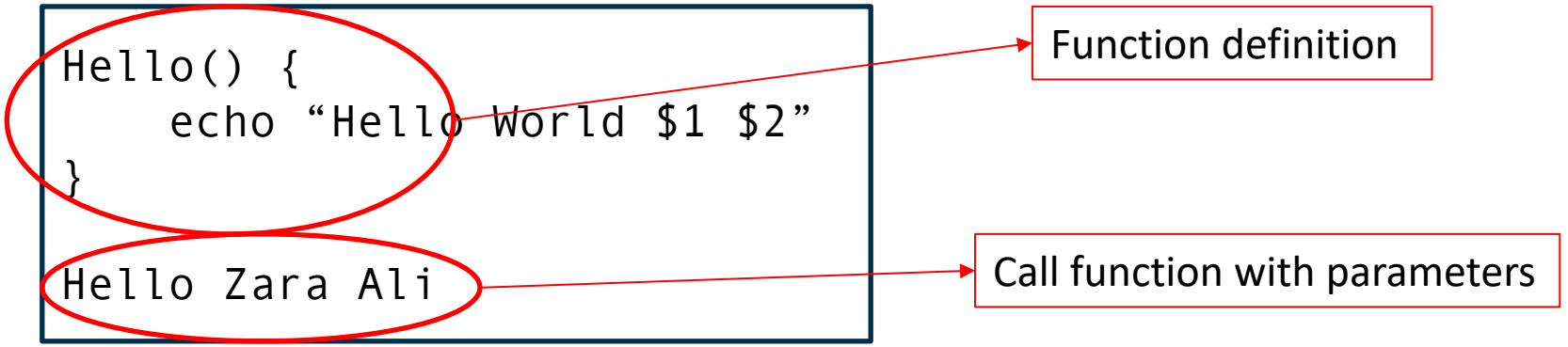
- Functions behave the same as commands
- The exit status of the function is the exit status of the last executed process
- Parameters are placed in variables `$1`, … , `$9`
- Use '`return`' to exit from the function early
- Use the '`local`' keyword to make local variables

```
Hello() {
    echo "Hello World $1 $2"
}

Hello Zara Ali
```

Functions

- Functions behave the same as commands
- The exit status of the function is the exit status of the last executed process
- Parameters are placed in variables `$1`, … , `$9`
- Use '`return`' to exit from the function early
- Use the '`local`' keyword to make local variables



Further reading

- The Bash Manual
  - www.gnu.org/software/bash/manual/bashref.html
- Advanced Bash-Scripting Guide
  - www.tldp.org/LDP/abs/html/
- UNIX tutorials
  - www.tutorialspoint.com/unix/

Exercises

- Blackboard
- Course webpage
  - http://msdl.cs.mcgill.ca/people/hv/teaching/ComputerSystemsArchitecture/#CS3