CS Essentials

Session 3: Bash Essentials 1

Introduction to Bash

What is the shell? - reminder

- the shell is the way in which you interact with your computer alongside with the commands you use
- simply put, it takes the commands you type and gives them to the OS to perform them

What is scripting and why use it?

- it means using the shell together with the text editor (maybe Vim?) to create a file containing the way in which you want to execute a certain set of commands

What is Bash?

- command language + shell
- it is an acronym for Bourne-again shell, where Bourne was another widely-used shell until the apparition of Bash
- everything we write in the command line can be written in a bash script and viceversa

Bash scripting

How do I write a bash script?

- open a file using Vim (say first_script) in the terminal and give it the extension .sh
- the first line of the file is:

$$\#!/bin/bash$$

- #! = Shebang (hash + bang)
- /bin/bash = the path to the interpreter that should be used to run the rest of the lines existing in the file
- note: formatting is essential; the command needs to be on the very first line of the file (not even an empty line can be before it)

First bash script

```
#!/bin/bash

echo"Hello, world!"
```

- echo - prints what you write after it on the screen
- save the script and run it : bash first_script.sh

Variables

What is a variable?

- a temporary location for a certain piece of information

What can you do with a variable?

1. You can set a variable
   - to set a variable, write the name of the variable and the information you want to store in it

     myVariable=Hello

   - Note: when setting a variable, you want to use descriptive names

Variables

2. You can read a variable
   - to read a variable, simply put the $ symbol in front of it (that means you are accessing the information that variable holds)

```
echo $myVariable
```

   - when you run this, Hello should appear on the screen

Variables

Special variables

- $0 - the name of the Bash script
- $1 - $9 - the first 9 arguments you give to the Bash script when running it
- $USER - the username of the user running the script

Using commands when setting variables

- to assign the output of a certain command to a variable, we just put the command inside brackets and add $ before it

variable=$(ls)

Variables

- when setting or reading a variable that has special characters, you need to use quotes (or escape every special character with backslash)

Single quotes vs double quotes

- single quotes will assign your variable exactly what you write between them
- double quotes will perform the substitutions before that

variable='Hello, $USER!' vs variable="Hello, $USER!"

Input

The read command

- we use the read command to ask the user for an input

```
echo Hi! What is your name?
read name
echo Nice to meet you, $name!
```

- read assigns the variable which you read to whatever the user types in
- read can be used with options, two common ones being -p(prompt) and -s(silent)

```
read -p "What is your username?" username
read -sp "what is your password?" userpass
```

Input

- read can also be used with more variables at a time
- it separates the inputs by splitting them on whitespaces

read -p "What are your friends' names?" friend1 friend2 friend3

echo "Your friends are $friend1, $friend2, $friend3!"

Arithmetic

1. Let - works in a few different ways:
   - let var1=1+2 (var1 now contains 3)
   - let "var2 = 1 + 2" (var2 now contains 3)
   - let var1++ (var1 now contains 4)
   - let "var3 = $var1 + $var2" (var3 now contains 7)
   - let "var4 = $var3 + $1" (var4 now contains 7 + the argument provided by the user)
   - we can use the basic operations, such as +, -, *, /, %, as well as ++ and −−, which are shortcuts for +1 and -1

Arithmetic

2. Expr
   - works in the same way as let, but prints the result instead of assigning it to a variable
   - you do not need to use quotes, but you must add spaces between the operands and the operators
   - you can store the value of the expression into a variable by using command substitution

Examples:

- expr 1 + 2 (this will print 3)
- expr 1+2 (this will print 1+2)
- var5=$(expr 1 + 2) (var5 now contains 3)

Arithmetic

3. Double parentheses
   - var6=$((1 + 2)) (var6 now contains 3)
   - var7=$((1+2)) (var7 now contains 3)
   - var8=$(($var6 + $var7)) (var8 now contains 6)
   - ((var8−−)) (var8 now contains 5)

If statements

How do they work?

- if statements are used to decide whether or not to execute a piece of code based on a certain condition

```
if [the condition to be met]
then
        commands
fi
```

- everything we write between then and fi will be executed if the initial condition is met

If statements

What conditions can we use?

- string1 = string2 - string1 is equal to string2
- string1 != string2 - string1 is not equal to string2
- integer1 -eq integer2 - integer1 is equal to integer2
- integer1 -gt integer2 - integer1 is greater than integer2
- integer1 -lt integer2 - integer1 is less than integer2
- !condition - condition is false
- -n string - the length of string is greater than 0
- -z string - the length of the string is 0
- -e file - file exists
- -d file - file exists and is a directory

If statements

- you can also check for permissions by putting -r, -w, -x in front of a filename
- Note: -eq is different from =. When we write var1 -eq var2, we check if the stringvar1 is the same as the string var2; = only checks the numberical value of the parameters we give (012 = 12 is true but 012 -eq 12 is not)

Example:

```
if [ $1 = $RANDOM ]
then
    That is a weird coincidence!
fi
```

If else statements

Structure:

```
if [ condition to be met ]
then
    commands1
else
    commands2
fi
```

- Note: indenting your code is useful especially when maintaining large pieces of code

If else statements

Example:

```
if [$RANDOM -gt $1]
then
    echo Your number is smaller than the random one!
else
    echo Your number is bigger than the random one!
fi
```

Nested if statements

```
myvar1=3
myvar2=100
if [ $myvar1 -lt $1 ]
then
    echo Your number is larger than $myvar1!
    if [ $myvar2 -lt $1 ]
    then
        echo Your number is also larger than $myvar2!
    else
        echo Your number is smaller than $myvar2!
    fi
fi
```

Elif statements

```
if [ condition1 ]
then
        commands1
elif [ condition2 ]
then
        commands2
else
        commands3
fi
```

Elif statements

Example:

```
if [ $age -gt 18 ]
then
        echo You can go out!
elif [ $answer = yes ]
then
        echo You can go out!
else
        echo You cannot go out!
fi
```

Combining conditions

And:

```
if [ condition1 ] && [ condition2 ]
then
    commands
fi
```

Or:

```
if [ condition1 ] || [ condition2 ] || [
condition3 ]
then
    commands
fi
```

Example

```
if [ -d $1 ] && [ -d $2 ]
then
    echo Both test1 and test2 are directories!
elif [ -d $1 ] || [ -d $2 ]
then
    echo Only one of the files is a directory.
else
    echo Neither of the files is a directory.
fi
```

Practice!

Thank you!