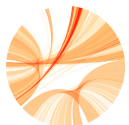


BIO609: Unix

Part 2: Bash scripting



With short code snippets and exercises



Universität
Zürich^{UZH}



gregor_rot

What is bash?



Bash is a **Unix shell** and **command language**

- default login shell
- **command processor** that runs in a text window (**console**)
- user types commands, which are executed
- **script file** also called a **shell script** (usually named .sh)
- **filename globbing** (wildcard matching, * ?)
- **piping** |
- **\$variables**
- **control** structures, **condition** testing, **iteration** (if, for)

```
root@kali:~# pwd
/root/.ssh
root@kali:~# cd /usr/portage/app-shells/bash
root@kali:~# cd /usr/portage/app-shells/bash & ls -al
total 120
drwxr-xr-x 3 portage portage 1024 Jul 25 18:06 .
drwxr-xr-x 30 portage portage 1024 Aug  7 22:39 ..
-rw-r--r-- 1 root root 5580 Jul 25 18:06 ChangeLog
-rw-r--r-- 1 root root 27892 Jul 25 18:06 Manifest
-rw-r--r-- 1 portage portage 4542 Mar 23 21:37 bash-3.1-p17.ebuild
-rw-r--r-- 1 portage portage 5977 Mar 23 21:37 bash-3.2-p39.ebuild
-rw-r--r-- 1 portage portage 6163 Mar  5 14:37 bash-3.2-p40.ebuild
-rw-r--r-- 1 portage portage 5988 Mar 23 21:37 bash-3.2-p48.ebuild
-rw-r--r-- 1 portage portage 2543 Mar  5 14:37 bash-4.0-p18.ebuild
-rw-r--r-- 1 portage portage 6230 Apr  5 14:37 bash-4.0-p18.ebuild
-rw-r--r-- 1 portage portage 5549 Mar 14 08:32 bash-4.0-p17.ebuild
```

My first bash script

```
$ vi first_script.sh
```

vi is in command mode
press **i** (enters **insert mode**)

```
#!/bin/bash  
echo "My first script :)"
```

press **ESC** (enters **command mode**)
type **":wq"**

```
$ chmod +x first_script.sh  
$ ./first_script.sh
```

set executable flag to first_script.sh
and run the script

```
bash script :)
```

output on the screen

Writing to a file

Special character **>** redirects output to a file

```
#!/bin/bash
#this is a comment
echo "Hello World" > log.txt
echo "How are you" > log.txt
```

→ writes "Hello World" to log.txt

→ overwrites log.txt with "How are you"

Special character **>>** appends output to a file

```
#!/bin/bash
#this is a comment
echo "Hello World" > log.txt
echo "How are you" >> log.txt
```

→ writes "Hello World" to log.txt

→ appends "How are you"

Variables

Variables are "named boxes" for values

- name=value (no spaces around =)
- to access, use \$ sign
- enclose variable name in {} if using in combination with other names
- some environment variables that are defined by the shell, like \$HOME

```
#!/bin/bash
home_folder=/home/gregor
echo "My home folder is $home_folder"
echo "My data folder is ${home_folder}/data"
```



Write a script that will store the environment variable \$HOME to the file home_folder.txt

Arrays

Arrays are simply variables with multiple values (lists)

- name=(value0 value1 value2 ...)
- access by index starting from 0

```
#!/bin/bash
color=(red green blue)
echo ${color[*]} # echo all values from array
color[3]=yellow # add new value to array
echo "The sky is ${color[2]}"
echo "There are ${#color[*]} colors in my array"
```

FOR loop

Iterate over values and repeat parts of code

```
#!/bin/bash
for i in 1 2 3 4 5
do
    echo "Current iteration is $i"
done

for i in {1..100}
do
    echo "Current iteration is $i"
done
```

FOR loop (more examples)

Iterate over values and repeat parts of code

```
#!/bin/bash
for i in *.sh
do
    echo "Current script file $i"
done

colors=(red green blue)
for color in ${colors[*]}
do
    echo "The color is $color"
done
```


FOR loop, a convenient list of strings

```
#!/bin/bash

list="item1
item2
item3
item4"

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

WHILE loop

Iterate until conditions are met

```
#!/bin/bash
count=0
while [[ $count -lt 4 ]]
do
    echo "Current count is $count"
    let count=count+1
done
```

IF statement

Check a condition and reach depending on the outcome

- can be composed of many **elif** statements
- always tested in order of appearance
- **else** clause is optional
- boolean operator to test if file exists: **-e**
- **string1 = string2**, **string1 != string2**

```
#!/bin/bash
if [[ something1 ]]
then
    command1
elif [[ something2 ]]
then
    command2
else
    command3
fi
```

```
#!/bin/bash
if [[ -e log.txt ]]
then
    echo "log file exists"
else
    echo "log does not exist"
```

```
#!/bin/bash
name=Andrej
if [[ $name = "Gregor" ]]
then
    echo "Your name is Gregor"
else
    echo "Andrej"
fi
```

IF cheat sheet

FILES conditions	-e FILE	does FILE exist?
STRING conditions	string1 = string2 string1 != string2	are strings equal? are strings different?
INTEGER conditions	int1 -eq int2 int1 -ne int2 int1 -lt int2 int1 -gt int2 int1 -le int2 int1 -ge int2 int1%int2	are numbers equal? are numbers not equal? is int1 < int2 ? is int1 > int2 ? is int1 <= int2 ? is int1 >= int2 ? division remainder (modulo, mod)
EXPRESSIONS	!exp exp1 && exp2 exp1 exp2	negates expression (logical NOT) logical AND logical OR

IF another example

```
#!/bin/bash
for i in {1..4}
do
    if [[ $i = 1 ]]
    then
        echo "first round"
    elif [[ $i = 2 ]]
    then
        echo "second round"
    else
        echo "round $i"
    fi
done
```



Write a script that will print out all numbers between 1 and 1000 that are dividable by 13.

```
$( (number%13) ) -eq 0
```



Write a script that will check if "black" is inside the color array we used before and will print "yes" if it's there

```
colors=(red green blue) # the color array
```

Commands in parallel

Run command in the background, add **&**

```
command1 &  
command2 &  
wait # if you want to wait until commands are done
```

Background commands in a for loop, with maximum number of threads

```
count=0  
for i in {1..10}  
do  
  (  
    command1  
    command2  
  ) &  
  let count+=1  
  
  # () for expression to be interpreted as mathematical operation  
  # % modulo, returns the remainder of the division  
  
  if [[ $((count%4)) -eq 0 ]] # max 4 threads  
  then  
    wait  
  fi
```

Special characters

#	comment character, anything after on same line is ignored
\$	expansion character (variables)
"text"	protects text from being split into multiple words or arguments
'text'	similar as "text", however prevents special characters meaning
\	escape character, prevents next character to be special
> and <	redirect characters (of input / output of a command)
	pipe character, sends output of one command to the input of the next



Write a script that will print out exactly **How are * doing \$today?**