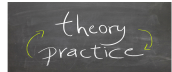
Unix/Linux Tutorial for Beginners
Session IX

Mihaela Martis

NBIS & Faculty of Medicine and Health Sciences
Division Cell Biology, IKE

Today's schedule



$9^{00} - 10^{30}$	*Shell scripting II*
$10^{30} - 10^{40}$	**Coffee break**
$10^{40} - 12^{10}$	*Exercises*
$12^{10} - 13^{10}$	**Lunch**
$13^{10} - 15^{00}$	*Exercises*
$15^{00} - 15^{20}$	**Coffee break**
$15^{20} - 17^{00}$	*Compute cluster (e.g. Uppmax)*

Conditionals in a bash script

- bash supports standard if...then...fi conditional/decision-making statement
- syntax:

```
if [conditional expression]
  then
    [if-statement]
  else
    [else-statement]
fi
```

- conditional expressions should be inside square braces with spaces around them

  correct: [[$a == $b]], incorrect: [[$a==$b]]
- [if-statement] executed if conditional expression true
- [else-statement] executed if conditional expression false

Forms of if...else statements

- if ... fi statement
- if ... else ... fi statement
- if ... elif ... else ... fi statement

Conditional expression & logical operators

- conditional expression returns 0 or 1
- it indicates the return value of the test and not success or error
- logical operators:
  - logical AND: &&
  - logical OR: ||
  - negation: !

Example conditional statement

```
#!/bin/bash

op1=4
op2=5
op3=10

sum=$(( $op1 + $op2 ))

if [[ ( $sum == $op3 ) && ( $op3 != $op1 ) ]]
  then
    echo "Bingo: the expression is true"
  else
    echo "the expression is false"
fi
```

Example conditional statement (II)

```
#!/bin/bash

if grep "pattern" myfile.txt > /dev/null &&
   grep "pattern" myfile2.txt > /dev/null
  then
    echo "found pattern in both files"
fi
```

- *grep* output is redirected to */dev/null* instead of the standard output

String and integer comparison operators

operator	description
-z str	empty string
$=$	identical strings
$!=$	different strings
-eq	values of 2 operands are equal
-ne	values of 2 operands differ
-gt	left value greater than right value
-lt	left value lower than right value
-ge	left value greater than or equal to the right value
-le	left value lower than or equal to the right value

Regular expression comparison ($=\sim$)

- the $=\sim$ operator takes a string on the left and an extended regular expression on the right
- returns 0 (success) if the regular expression matches the string, otherwise 1 (failure)

```
text="I'm free to do what I want!"
if [[ $text =~ "to do" ]]
  then
    echo "$text"
  else
    echo "It's not what I'm looking for"
fi
```

Metacharacters of regular expression

metacharacter	description
^	caret symbol to match the beginning of a line
$	matches end of the line
.	matches any single character
*	matches zero or more of the preceding character
?	matches zero or one of the preceding character
\n	matches a newline character
\t	matches a tab character
\s	matches any whitespace character
\w	matches any word character including underscore
\d	matches a digit character

File and directory test operators

operator	description
-f file	checks if file is an ordinary file
-e file	checks if file exists
-d dir	checks if file is a directory
-r file	checks if file is readable
-w file	checks if file is writable
-x file	checks if file is executable

```
if [ -f myfile.txt ]
  then
    [...]
fi
```

Processing files with bash loops

- aim: apply the same pipeline on multiple files
- 3 essential parts to create a pipeline to process a set of files
  - select which files to apply the commands to
  - loop over the data and apply the commands
  - keep track of the names of any created output file

The while-loop

```
while command1
do
  Statement1
  Statement2
done
```

The while-loop

```
while command1
do
  Statement1
  Statement2
done
```

```
#!/bin/bash

a=0
while [ $a -lt 10 ]
do
  b=$a
  a=$(expr $a + 1)
done

echo "a=$a"
echo "b=$b"
```

$\rightarrow$ a = 10, b = 9

Read files line by line

- syntax:

```
while read line
do
  COMMAND
done <input.file
```

- examples:

```
#!/bin/bash
file="$HOME/data.txt"
while read line
do
  echo $line
done <$file
```

The for-loop

- it's used to cycle through an input one item at a time
- it splits each line when it sees any whitespace

```
for var in word1 word2 ... wordn
do
  statements
done
```

The for-loop

- it's used to cycle through an input one item at a time
- it splits each line when it sees any whitespace

```
for var in word1 word2 ... wordn
do
  statements
done
```

```
for file in *.fastq
do
  echo "working on $file!"
done


for i in 1 2 3
do
  echo -e $i^2 = $(( $i * $i ))
done
```

Summary

- shell scripts help automating workflows and applying them on multiple data sets
- shell scripts should be robust and reproducible
- if statements & loops
- logical & comparison operators