



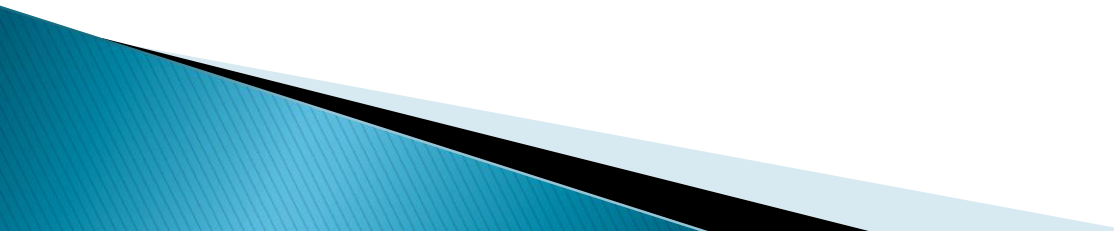
Informatics on High-throughput Sequencing Data

(Summer Course 2020)

Day 8



Agenda

- ▶ **Shell Scripting**
 - ▶ **Variables**
 - ▶ **User Inputs**
 - ▶ **Arithmetic**
 - ▶ **Functions**
 - ▶ **IF statements**
 - ▶ **Loops**
- 

Numeric and String Comparisons

Bash Shell Numeric and String Comparisons

Description	Numeric Comparison	String Comparison
less than	-lt	<
greater than	-gt	>
equal	-eq	=
not equal	-ne	!=
less or equal	-le	N/A
greater or equal	-ge	N/A
Shell comparison example:	[100 -eq 50]; echo \$?	["GNU" = "UNIX"]; echo \$?

- **and** - &&
- **or** - ||

<https://linuxconfig.org/bash-scripting-tutorial-for-beginners>

Numeric and String Comparisons

- ▶ we use square brackets and numeric comparison operators to perform the evaluation.
 - ▶ Using `echo $?` command, we check for a return value of the previously executed command.
 - ▶ If the return value is equal to `0`, then the comparison evaluation is true. However, if the return value is equal to `1`, the evaluation resulted as false.
-
- ▶ `a=4`
 - ▶ `b=8`
 - ▶ `[$a -lt $b]`
 - ▶ `echo $?`

Numeric and String Comparisons

Operator	Description
! EXPRESSION	The EXPRESSION is false.
-n STRING	The length of STRING is greater than zero.
-z STRING	The length of STRING is zero (ie it is empty).
STRING1 = STRING2	STRING1 is equal to STRING2
STRING1 != STRING2	STRING1 is not equal to STRING2
INTEGER1 -eq INTEGER2	INTEGER1 is numerically equal to INTEGER2
INTEGER1 -gt INTEGER2	INTEGER1 is numerically greater than INTEGER2
INTEGER1 -lt INTEGER2	INTEGER1 is numerically less than INTEGER2
-d FILE	FILE exists and is a directory.
-e FILE	FILE exists.
-r FILE	FILE exists and the read permission is granted.
-s FILE	FILE exists and its size is greater than zero (ie. it is not empty).
-w FILE	FILE exists and the write permission is granted.
-x FILE	FILE exists and the execute permission is granted.

<https://ryanstutorials.net/bash-scripting-tutorial/bash-if-statements.php>

Numeric and String Comparisons

- ▶ `=` is slightly different to `-eq`. `[001 = 1]` will return false as `=` does a string comparison (ie. character for character the same) whereas `-eq` does a numerical comparison meaning `[001 -eq 1]` will return true.
- ▶ `[]` is just a reference to the command `test`.

```
Terminal
1. user@bash: test 001 = 1
2. user@bash: echo $?
3. 1
4. user@bash: test 001 -eq 1
5. user@bash: echo $?
6. 0
7. user@bash: touch myfile
8. user@bash: test -s myfile
9. user@bash: echo $?
10. 1
```

If statements

```
if [ <some test> ]  
then  
    <commands>  
fi
```

if_example.sh

```
1. #!/bin/bash  
2. # Basic if statement  
3.  
4. if [ $1 -gt 100 ]  
5. then  
6.     echo Hey that's a large number.  
7.     pwd  
8. fi  
9.  
10. date
```

<https://linuxconfig.org/bash-scripting-tutorial-for-beginners>

Nested If statements

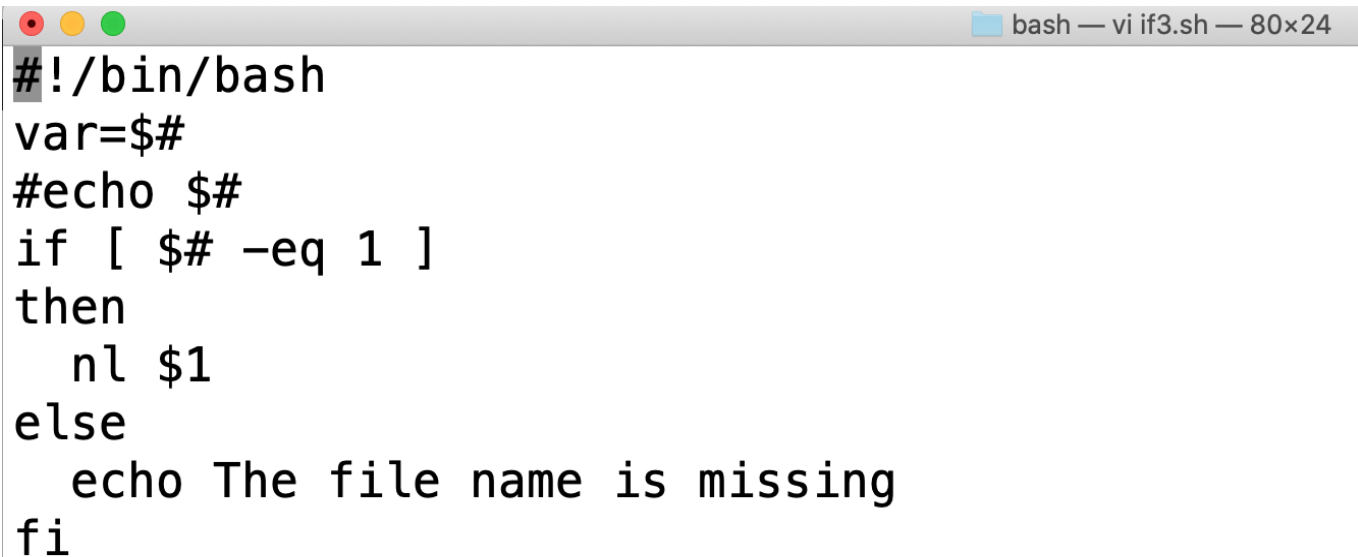
nested_if.sh

```
1. #!/bin/bash
2. # Nested if statements
3.
4. if [ $1 -gt 100 ]
5. then
6.     echo Hey that\'s a large number.
7.
8.     if (( $1 % 2 == 0 ))
9.     then
10.         echo And is also an even number.
11.     fi
12. fi
```

<https://linuxconfig.org/bash-scripting-tutorial-for-beginners>

If Else

```
if [ <some test> ]  
then  
    <commands>  
else  
    <other commands>  
fi
```



A terminal window titled "bash — vi if3.sh — 80x24" displays a bash script. The script starts with a shebang line, sets a variable, and uses an if-else statement to check if the number of arguments is 1. If true, it lists the arguments; otherwise, it prints a message about a missing file name.

```
#!/bin/bash  
var=$#  
#echo $#  
if [ $# -eq 1 ]  
then  
    nl $1  
else  
    echo The file name is missing  
fi
```

<https://linuxconfig.org/bash-scripting-tutorial-for-beginners>

If Elif Else

if_elif.sh

```
1. #!/bin/bash
2. # elif statements
3.
4. if [ $1 -ge 18 ]
5. then
6.     echo You may go to the party.
7. elif [ $2 == 'yes' ]
8. then
9.     echo You may go to the party but be back before midnight.
10. else
11.     echo You may not go to the party.
12. fi
```

```
if [ <some test> ]
then
    <commands>
elif [ <some test> ]
then
    <different commands>
else
    <other commands>
fi
```

Boolean Operations

and.sh

```
1. #!/bin/bash
2. # and example
3.
4. if [ -r $1 ] && [ -s $1 ]
5. then
6.     echo This file is useful.
7. fi
```

or.sh

```
1. #!/bin/bash
2. # or example
3.
4. if [ $USER == 'bob' ] || [ $USER == 'andy' ]
5. then
6.     ls -alh
7. else
8.     ls
9. fi
```

Case Statements

case.sh

```
1. #!/bin/bash
2. # case example
3.
4. case $1 in
5.     start)
6.         echo starting
7.         ;;
8.     stop)
9.         echo stoping
10.        ;;
11.    restart)
12.        echo restarting
13.        ;;
14.    *)
15.        echo don\'t know
16.        ;;
17. esac
```

```
case <variable> in
    <pattern 1>
        <commands>
        ;;
    <pattern 2>
        <other commands>
        ;;
esac
```

References

- ▶ <https://bioinformatics.uconn.edu/unix-basics/#>
- ▶ <https://learn.gencore.bio.nyu.edu/ngs-file-formats/quality-scores/>
- ▶ <https://coding4medicine.com/Members/pages/home/>
- ▶ <https://open.oregonstate.education/computationalbiology/chapter/patterns-regular-expressions/>
- ▶ <https://bioinformaticsworkbook.org/Appendix/Unix/unix-basics-3grep.html#gsc.tab=0>
- ▶ <https://datacarpentry.org/shell-genomics/04-redirection/>
- ▶ https://www.hadriengourle.com/tutorials/command_line/
- ▶ http://people.duke.edu/~ccc14/duke-hts-2018/cliburn/Bash_in_Jupyter.html
- ▶ <https://github.com/tavareshugo/BioPipelines>
- ▶ RNA-Seq, https://hbctraining.github.io/Intro-to-rnaseq-hpc-02/lessons/02_assessing_quality.html
- ▶ Exam, <http://hpc.ilri.cgiar.org/beca/bioinfo/linux.html>
- ▶ <http://hpc.ilri.cgiar.org/beca/bioinfo/index.html#BSA>
- ▶ <http://userweb.eng.gla.ac.uk/umer.ijaz/bioinformatics/linux.html>
- ▶ https://wiki.bits.vib.be/index.php/Linux_command_line
- ▶ Good to continue: <https://www.hadriengourle.com/wrangling-genomics/aio/>

Thanks!

// | ?