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 $?
```

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

Numeric and String Comparisons

Operator	Description
! EXPRESSION	The EXPRESSION is false.
-n STRING	The length of STRING is greater than zero.
-z STRING	The lengh 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 it's 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

```
user@bash: test 001 = 1
user@bash: echo $?
1
user@bash: test 001 -eq 1
user@bash: echo $?
0
user@bash: touch myfile
user@bash: test -s myfile
user@bash: echo $?
1
```

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

If stataments

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

if_example.sh

```
#!/bin/bash
# Basic if statement

if [ $1 -gt 100 ]
then
   echo Hey that\'s a large number.
   pwd
fi

date
```

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

Nested If statements

nested_if.sh

```
#!/bin/bash
# Nested if statements

if [ $1 -gt 100 ]
then
   echo Hey that\'s a large number.

   if (( $1 % 2 == 0 ))
   then
      echo And is also an even number.
   fi
fi
```

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

If Else

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

bash — vi if3.sh — 80×24

```
#!/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

```bash
#!/bin/bash
# elif statements

if [ $1 -ge 18 ]
then
   echo You may go to the party.
elif [ $2 == 'yes' ]
then
   echo You may go to the party but be back before midnight.
else
   echo You may not go to the party.
fi
```

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

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

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
```

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

Case Statements

case.sh

```bash
#!/bin/bash
# case example

case $1 in
   start)
      echo starting
      ;;
   stop)
      echo stoping
      ;;
   restart)
      echo restarting
      ;;
   *)
      echo don\'t know
      ;;
esac
```

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

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

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-O2/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!

// | ?