

Introduction to **Unix command line** & **Shell programming**

Wenbin Guo
Bioinformatics IDP, UCLA
wbguo@ucla.edu
2022 Spring

Notations of the slides

- Code chunk starts with "➤", e.g.
➤ echo 'Hello world!'
- Link is underlined
- Practice comes with



Agenda

- Day 0: Preparation
- Day 1: Introduction to Unix Command line
 - Navigating through file system
 - Manipulating files and directories
- Day 2: Advanced Command line
 - Useful commands for file processing
 - sed, awk, grep, and regular expression
- Day 3: **Shell scripting** and running jobs on hoffman2



Day 3: Shell scripting and running jobs on hoffman2

Wenbin Guo
Bioinformatics IDP, UCLA
wbguo@ucla.edu
2022 Spring

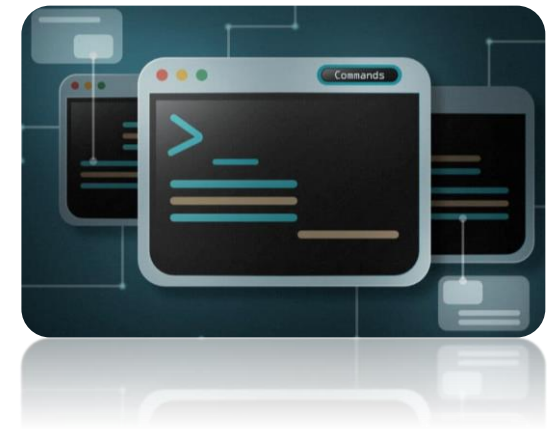
Overview

Time

- 3-hour workshop (45min + 45min + 30min + practices/Q&A)

Topics

- ☐ Variables and operations
- ☐ Condition statement
- ☐ Loop structure
- ☐ Submit jobs to hoffman2 cluster
- ☐ QUIZ will be released today
- ☐ Assignment will be released on Friday



Questions can be put into this [Google doc](#)

Summary-Day1&2

We have knowledge about

- Unix **file system** and how to **navigate** through it
- **Manipulating** the files and directories
- Advanced Command lines for **file processing**
- **sed**, **awk**, **grep** for contents editing/extracting/searching
- Regular expression (**RegEx**)

And we have experience in

- **Accessing** hoffman2 cluster & **file transfer**
- **Requesting resources** on hoffman2
- Executing **commands** and using **pipe**



Summary-Day1&2

Commands

Navigation

Command	Function
ssh	Connect to hoffman2
exit	Close the remote connection
pwd	Print working directory
cd	Change directory
ls	List the information of files/folder (note the flag's usage)

File/directory Manipulation

Command	Function
mkdir	Create a directory
touch	Create a file / change the time stamp
rm	Remove file or directory
cp	Copy file or directory
mv	Move file or directory
cat	Print the file contents to standard output
vi	Modify a file / create a file
chmod	Change file permission
gzip	Compress/decompress a file
tar	Create a compressed archive
scp/sftp	File transfer
man/--help	Get help

Summary-Day1&2

Commands

Command	Function
cat	Show file contents/merge files
more/less	Show file contents
head/tail	Show file contents at beginning/end
zcat/zmore	Show contents of compressed file
sort	Sort a file based on column
>/>>	Redirect output to files (write/append)
	Combine commands
wc	count
cut	Extract a part of string
uniq	Show the unique line
diff	Show the difference of 2 files
sed	Non-interactive editing
awk	Text processing
grep	search string/pattern in the file

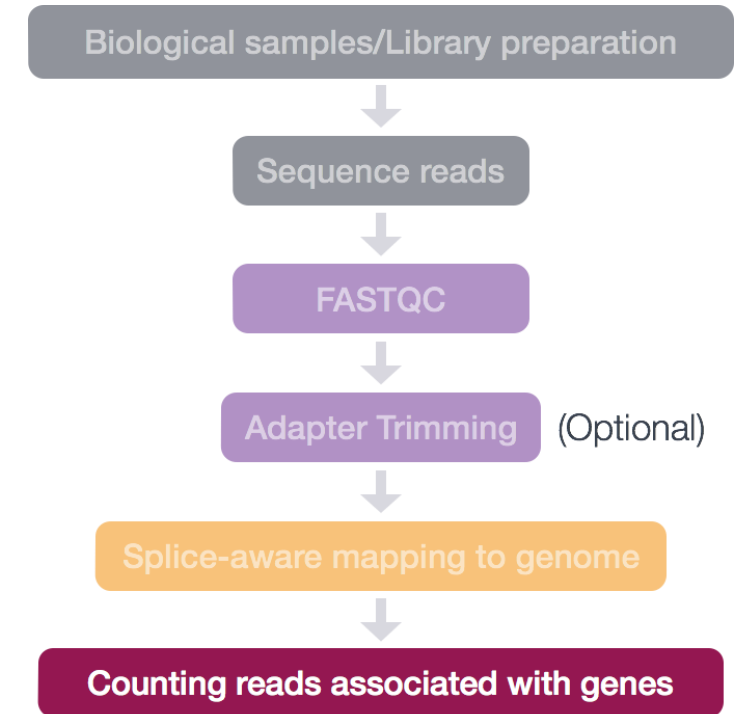
File processing

Command line is great, but...

- A task might consist of **multiple steps**
 - We want to run multiple commands at one pass
- A task might be done **repeatedly**
 - We don't want to type in the commands over and over again

We want sth to automate the tasks!

(an example of work hard and **work smart**)



RNA-seq workflow example

Shell scripting

- Create a `.sh` file using text editor tool (**vi**, nano, etc.)
- Write down your commands and save it to the file
- Execute the commands in the file

The script can be executed whenever it's needed, so that you **avoid the repetitive typing work!**

Preparation

- Connect to hoffman2

- `ssh username@hoffman2.idre.ucla.edu`

- Request a interactive node

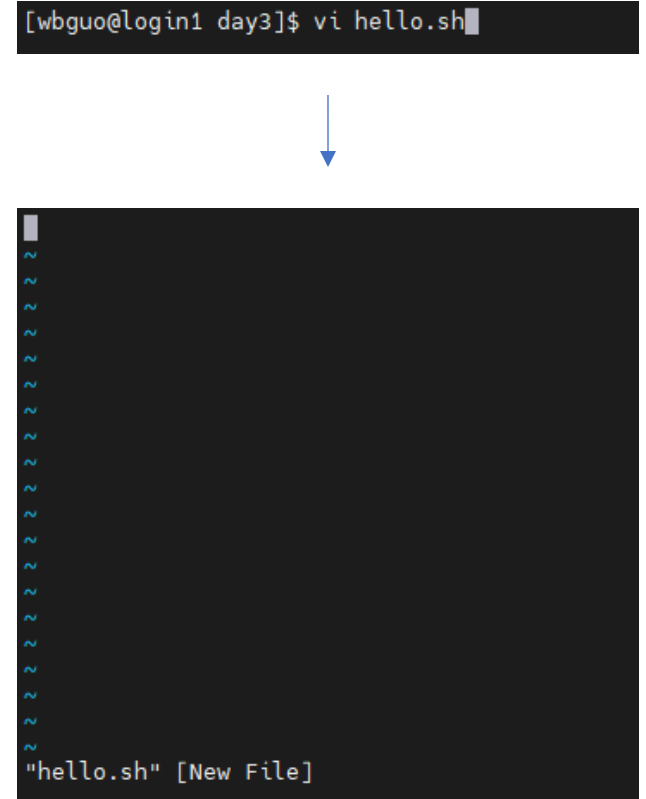
- `qssh -l h_rt=03:00:00,h_data=256M`

- Get the working materials (if you don't have them)

- `git clone https://github.com/wbvguo/qcbio-intro2Unix.git`

Editing file with vi

- Press "**i**" to type in new text (**i**nsertion mode)
- Press "**Esc**" to jump out the insertion mode
- Enter
 - **:w** (**w**rite)
 - **:q** (**q**uit)
 - **:wq** (**w**rite and **q**uit)
 - **:q!** (force **q**uit – the modification will not be saved)



The image shows a terminal window with the command `[wbguo@login1 day3]$ vi hello.sh` at the top. A blue arrow points down from the terminal to a screenshot of the vi editor interface. The editor has a dark background with a light blue vertical line on the left side. The status bar at the bottom shows `"hello.sh" [New File]`.

Useful commands

- **echo**: pass the input string to standard output (display messages)
- Syntax: `echo -<flag> <your string>`

```
[wbguo@login1 day3]$ echo 'Hello world!'
Hello world!
```

- Flag options:
 - **-n**: do not output the trailing new line
 - **-e**: enable interpretation of backslash escapes

```
[wbguo@login1 day3]$ echo 'The current time is:'; date
The current time is:
Thu Sep 30 01:51:45 PDT 2021
[wbguo@login1 day3]$ echo -n 'The current time is:'; date
The current time is:Thu Sep 30 01:51:49 PDT 2021
```

Tips:

- `date` will output the current system time
- To run 2 separate commands in one line: `command1;command2`

Writing your first shell script

➤ `vi hello.sh`

- Press "**i**", type in the following contents

```
#!/bin/bash
# print hello
echo "Hello world!"
echo -n "From: "
echo $USER
```

When creating a shell script file, you should **specify the shell** you are using in the **first line** of the file

Comments will **not be executed**
(Adding comments can usually help you understand the code)

Code lines to be executed

- After that, press "**Esc**" and input **`:wq`**

Executing your first shell script

- Option 1: use the `bash` command

- `bash hello.sh`

```
#!/bin/bash

# print hello

echo "Hello world!"
echo -n "From: "
echo $USER
```



```
[wbguo@login1 day3]$ bash hello.sh
Hello world!
From: wbguo
```

- Option 2: grant execution permission to `hello.sh`

- `chmod u+x hello.sh`

- `./hello.sh`

Note: You can also use the [absolute path](#) to execute the file

```
[wbguo@login1 day3]$ ls -l hello.sh
-rw-r--r--. 1 wbguo matteop 77 Sep 30 02:05 hello.sh
[wbguo@login1 day3]$ chmod u+x hello.sh
[wbguo@login1 day3]$ ls -l hello.sh
-rwxr--r--. 1 wbguo matteop 77 Sep 30 02:05 hello.sh
[wbguo@login1 day3]$ ./hello.sh
Hello world!
From: wbguo
```

Practice:



- Create the `hello.sh` file according to the previous slides
- Practice the 2 types of execution

Variables

- Environment variables
- User variables

Advanced contents

- Perform operation on variables
- Assign command output to variables
- Pass parameters from command line to scripts

Variables - environment

- Environment variables

- USER
- HOSTNAME
- HOME
- PWD
- BASH
- PATH
- ...

```
[wbguo@login1 day3]$ echo $USER
wbguo
[wbguo@login1 day3]$ echo $HOSTNAME
login1
[wbguo@login1 day3]$ echo $HOME
/u/home/w/wbguo
[wbguo@login1 day3]$ echo $PWD
/u/home/w/wbguo/qcbio-intro2Unix/day3
[wbguo@login1 day3]$ echo $BASH
/bin/bash
[wbguo@login1 day3]$ echo $PATH
/u/systems/UGE8.6.4/bin/lx-amd64:/u/local/apps/vcftools/0.1.16/gcc-4.8.5/bin:/u/local/apps/bowtie2/2.4.2:/u/local/apps/bedtools/2.30.0/gcc-4.8.5/usr/local/bin:/u/local/apps/samtools/1.11/gcc-4.8.5/bin:/u/local/apps/htslib/1.11/gcc-4.8.5/bin:/u/local/apps/bcftools/1.11/gcc-4.8.5/bin:/u/local/apps/R/4.1.0/gcc-4.8.5_MKL-2020/bin:/u/local/apps/curl/7.70.0/gcc-4.8.5/bin:/u/local/apps/python/3.7.3/gcc-4.8.5/bin:/u/local/apps/perl/5.32.1/bin:/u/local/apps/julia/1.6.3/bin:/u/local/apps/java/jre1.8.0_281/bin:/u/local/compilers/intel/2020.4/compilers_and_libraries_2020.4.304/linux/bin/intel64:/u/local/compilers/intel/2020.4/compilers_and_libraries_2020.4.304/linux/bin:/u/local/compilers/intel/2020.4/compilers_and_libraries_2020.4.304/linux/mpi/intel64/libfabric/bin:/u/local/compilers/intel/2020.4/compilers_and_libraries_2020.4.304/linux/mpi/intel64/bin:/u/local/compilers/intel/2020.4/debugger_2020/gdb/intel64/bin:/u/systems/UGE8.6.4/bin/lx-amd64:/u/local/bin:/u/local/sbin:/u/local/Modules/4.7.0/gcc-4.8.5/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/u/home/w/wbguo/apps:/u/home/w/wbguo/apps/blast-2.11.0+/bin:/u/home/w/wbguo/apps/cgmaptools-0.1.2:/u/home/w/wbguo/apps/FastQC-0.11.9:/u/home/w/wbguo/apps/TrimGalore-0.6.6:/u/home/w/wbguo/apps/Trimmomatic-0.39/
[wbguo@login1 day3]$
```

- Tips:

- To use the variable, simply put a **\$** before the variable name (reference to variables)
- Use **echo** to print the value of variables
- Use **set** to check all the environment variables

Scope: Environment variable can be referenced in all shells (global variable)

A little bit more about PATH

- Questions: how does the Unix system recognize my command?

```
[wbguo@login1 day3]$ date  
Thu Sep 30 02:23:10 PDT 2021  
[wbguo@login1 day3]$ dattt  
bash: dattt: command not found
```

A little bit more about PATH

- Questions: how does the Unix system recognize my command?

```
[wbguo@login1 day3]$ date
Thu Sep 30 02:23:10 PDT 2021
[wbguo@login1 day3]$ dattt
bash: dattt: command not found
```

- It will search the command in the PATH

```
[wbguo@login1 day3]$ which date
/usr/bin/date
[wbguo@login1 day3]$ which dattt
/usr/bin/which: no dattt in (/u/systems/UGE8.6.4/bin/lx-amd64:/u/local/apps/vcftools/0.1.16/gcc-4.8.5/bin:/u/local/apps/bowtie2/2.4.2:/u/local/apps/bedtools/2.30.0/gcc4.8.5/usr/local/bin:/u/local/apps/samtools/1.11/gcc-4.8.5/bin:/u/local/apps/htslib/1.11/gcc-4.8.5/bin:/u/local/apps/bcftools/1.11/gcc-4.8.5/bin:/u/local/apps/R/4.1.0/gcc-4.8.5_MKL-2020/bin:/u/local/apps/curl/7.70.0/gcc-4.8.5/bin:/u/local/apps/python/3.7.3/gcc-4.8.5/bin:/u/local/apps/perl/5.32.1/bin:/u/local/apps/julia/1.6.3/bin:/u/local/apps/java/jre1.8.0_281/bin:/u/local/compiler/intel/2020.4/compiler_and_libraries_2020.4.304/linux/bin/intel64:/u/local/compiler/intel/2020.4/compiler_and_libraries_2020.4.304/linux/bin:/u/local/compiler/intel/2020.4/compiler_and_libraries_2020.4.304/linux/mpi/intel64/libfabric/bin:/u/local/compiler/intel/2020.4/compiler_and_libraries_2020.4.304/linux/mpi/intel64/bin:/u/local/compiler/intel/2020.4/debugger_2020/gdb/intel64/bin:/u/systems/UGE8.6.4/bin/lx-amd64:/u/local/bin:/u/local/sbin:/u/local/Modules/4.7.0/gcc-4.8.5/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/u/home/w/wbguo/apps:/u/home/w/wbguo/apps/blast-2.11.0+/bin:/u/home/w/wbguo/apps/cgmaptools-0.1.2:/u/home/w/wbguo/apps/FastQC-0.11.9:/u/home/w/wbguo/apps/TrimGalore-0.6.6:/u/home/w/wbguo/apps/Trimmomatic-0.39/)
```

"which **command**" will return the path of executable file to the standard output

```
[wbguo@login1 day3]$ ls -l /usr/bin/date
-rwxr-xr-x. 1 root root 62200 Nov 16 2020 /usr/bin/date
```

Variables – user

- User variables
- Syntax: `var_name=value`
 - `Var1=10` Numerical
 - `var1="apple"` String
 - `var2=(10 20 30)` Array
- Note:
 - **No space** is allowed between the `var_name`, `=`, and `value`
 - Space is the default Field Separator in shell
 - The `var_name` is **case sensitive**
 - `Var1` and `var1` are 2 different variable names
 - Use **\$** to reference (`$var_name`, or `${var_name}`)
 - Avoid giving `var_name` an environment variable name
 - Data type is automatically assigned based on the value

Practice:



- print HOME, USER, PWD, PATH environment variable
- Create a user variable named var1, assign value 10, and print it to standard output
- Try the following command
 - `var2= 10`
 - `echo $var2`

Operations on variables

Operation on variables in shell is limited, but still possible

- Numeric operation (+, -, *, /, %):
 - Syntax: `$(operation)`
 - e.g. `var2=$((var1*2))`

Note, the output will only be int (not float)

Operations on variables

Operation on variables in shell is limited, but still possible

- Numeric operation (+, -, *, /, %):
 - Syntax: `$(operation)`
 - e.g. `var2=$((var1*2))`

Note, the output will only be int (not float)

```
[wbguo@login1 day3]$ var1=10
[wbguo@login1 day3]$ var2=$((var1*2))
[wbguo@login1 day3]$ echo $var2
20
```


Operations on variables

Operation on variables in shell is limited, but still possible

- Numeric operation (+, -, *, /, %):
 - Syntax: `$(operation)`
 - e.g. `var2=$((var1*2))`
- String operation:
 - Syntax: `$(expr operation)`
 - e.g. `var2=$(expr length "BANANA")`

Operator	Description
match STRING REGEXP	Returns the pattern match if REGEXP matches a pattern in STRING
substr STRING POS LENGTH	Returns the substring LENGTH characters in length, starting at position POS (starting at 1)
index STRING CHARS	Returns position in STRING where CHARS is found; otherwise, returns 0
length STRING	Returns the numeric length of the string STRING
+ TOKEN	Interprets TOKEN as a string, even if it's a keyword
(EXPRESSION)	Returns the value of EXPRESSION

expr can also perform numeric operation, for more information , please check the [bible book](#)

Operations on variables

Operation on variables in shell is limited, but still possible

- Numeric operation (+, -, *, /, %):
 - Syntax: `$(operation)`
 - e.g. `var2=$((var1*2))`
- String operation:
 - Syntax: `$(expr operation)`
 - e.g. `var2=$(expr length "BANANA")`

```
[wbguo@login1 day3]$ var2=$(expr length "BANANA")  
[wbguo@login1 day3]$ echo $var2  
6
```

expr can also perform numeric operation, for more information , please check the [bible book](#)

Operations on variables

Operation on variables in shell is limited, but still possible

- Numeric operation (+, -, *, /, %):
 - Syntax: `$(operation)`
 - e.g. `var2=$((var1*2))`
- String operation:
 - Syntax: `$(expr operation)`
 - e.g. `var2=$(expr length "BANANA")`
- Array operation:
 - Reference to the n-th element: `${array[n-1]}`
 - Print out the whole array: `${array[@]}`

Operations on variables

Operation on variables in shell is limited, but still possible

- Numeric operation (+, -, *, /, %):
 - Syntax: `$(operation)`
 - e.g. `var2=$((var1*2))`
- String operation:
 - Syntax: `$(expr operation)`
 - e.g. `var2=$(expr length "BANANA")`
- Array operation:
 - Reference to the n-th element: `${array[n-1]}`
 - Print out the whole array: `${array[@]}`

```
[wbguo@login1 day3]$ array=(10 20 30)
[wbguo@login1 day3]$ echo $array
10
[wbguo@login1 day3]$ echo ${array[@]}
10 20 30
[wbguo@login1 day3]$ echo ${array[2]}
30
```

Practice:



- print HOME, USER, PWD, PATH environmental variable
- Create a user variable named var1, assign value 10, and print it to standard output
- Try the following command
 - `var2= 10`
 - `echo $var2`
- Create a variable named var2, assign value 1, create another variable named var3, assign its value as `var2/var1`, what do you find?
- Try the array operation in the previous slide

Assign command output to variables

- Option1: use the backtick character (`)

➤ `var_name=`ls *``

- Option2: use `$()` format

➤ `var_name=$(ls *)`

```
[wbguo@login3 day3]$ ls
example_sh  jobs  qjob.sh  sjob.sh  test_break_continue.txt
[wbguo@login3 day3]$ array=`ls *job*`
[wbguo@login3 day3]$ echo ${array[@]}
jobs qjob.sh sjob.sh
[wbguo@login3 day3]$ array=$(ls *job*)
[wbguo@login3 day3]$ echo ${array[@]}
jobs qjob.sh sjob.sh
[wbguo@login3 day3]$
```

Pass parameters from command line to scripts

- Parameters' meaning
 - \$0: the script name
 - \$1: the first parameter
 - \$2: the second parameter
 - ...

```
#!/bin/bash

# accept parameters, store in variables
script_name=$0
para1=$1
para2=$2

echo "1st parameter is $para1"
echo "2nd parameter is $para2"
echo "script name is $script_name"
```

```
[wbguo@login1 day3]$ bash pass_para.sh apple banana
1st parameter is apple
2nd parameter is banana
script name is pass_para.sh
```

Practice:



- List the file/folder names in day3 folder and assign it to a variable named array, then print all the elements in array
- Write a script that can accept file name, and print out the file name and the file contents, test it on jobs under day3 folder

Condition statement

- if-then
 - if-then-else
 - Nested if-then
 - Conditions in if-then statement
-
- Example
 - If tomorrow is **sunny**, then I will **go to the beach**
 - If tomorrow is **sunny**, then I will **go to the beach**; **otherwise** I will **stay home**
 - If tomorrow is **sunny**, then I will **go to the beach**; if it's **rainy**, I will **wash my car**; **otherwise**, I will **stay home**

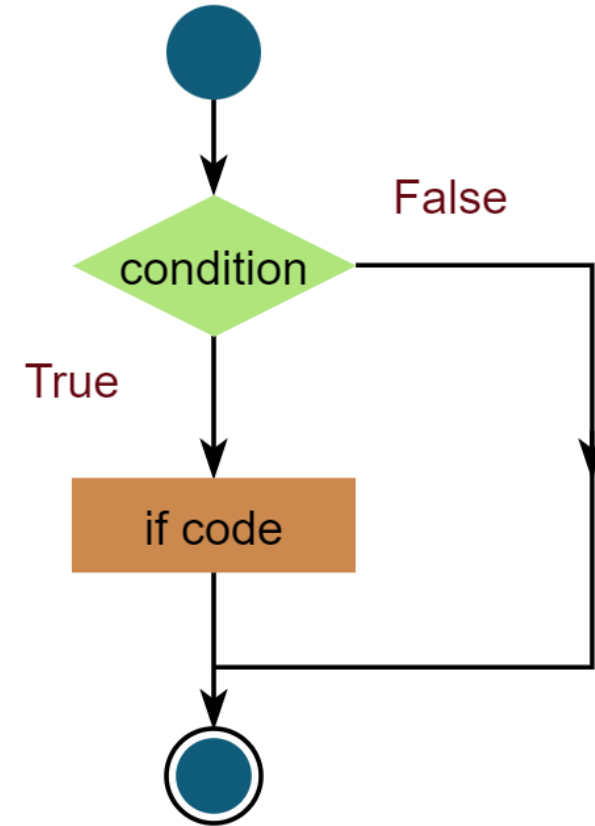
condition and **action**

if-then condition

- Syntax

```
if [ condition ]  
then  
    commands  
fi
```

Note: there must be a **space** after "[" and a **space** before "]", otherwise you will get an error



if-then condition

- Syntax

```
if command
then
    commands
fi
```

Usually the command after if is a condition statement in [], in the form of [condition]

Note: there must be a space after "[" and a space before "] ", otherwise you will get an error

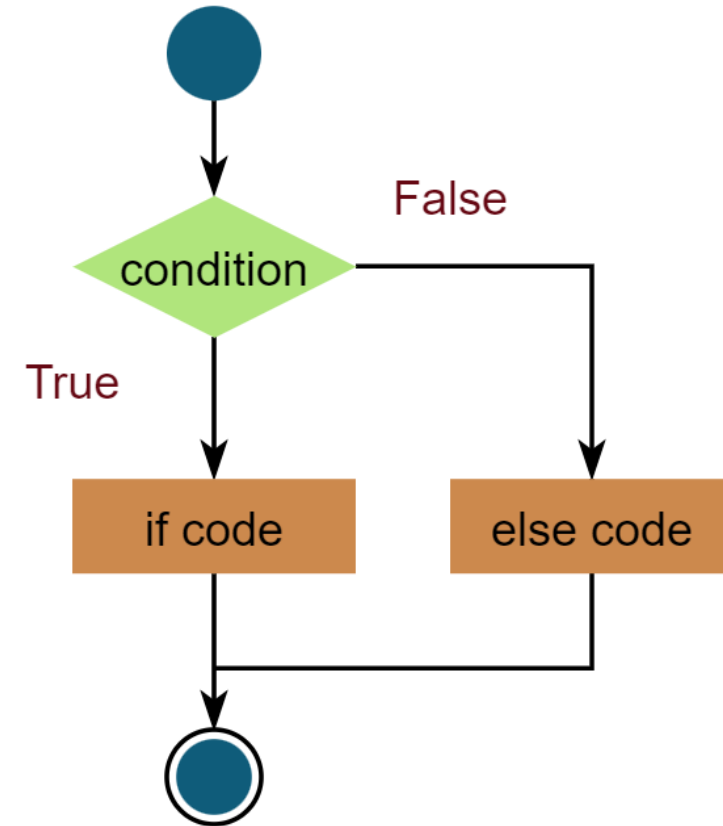
```
#!/bin/bash
# if then
var=1
if [ $var -le 2 ]
then
    echo "the value is less equal than 2"
fi
```

```
[wbguo@login1 day3]$ bash if_then.sh
the value is less equal than 2
```

if-then-else condition

- Syntax

```
if command
then
    commands
else
    commands
fi
```



if-then-else condition

- Syntax

```
if command
then
    commands
else
    commands
fi
```

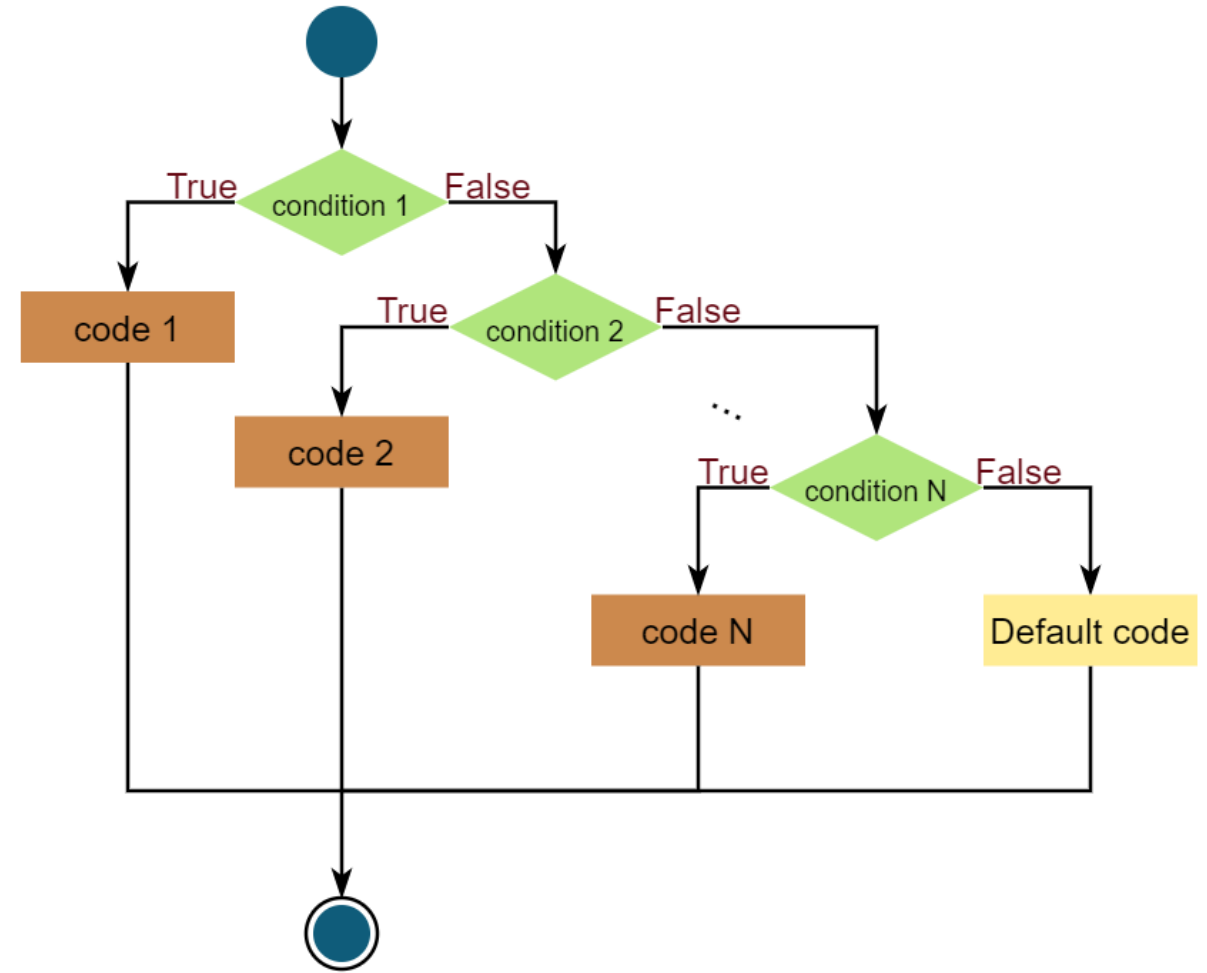
```
#!/bin/bash
# if then else
var=3
if [ $var -le 2 ]
then
    echo "the value is less equal than 2"
else
    echo "the value is greater than 2"
fi
```

```
[wbguo@login1 day3]$ bash if_then_else.sh
the value is greater than 2
```

Nested if-then condition

- Syntax

```
if command1
then
    command set 1
elif command2
then
    command set 2
elif command3
then
    command set 3
elif command4
then
    command set 4
fi
```



Nested if-then condition

- Syntax

```
if command1
then
    command set 1
elif command2
then
    command set 2
elif command3
then
    command set 3
elif command4
then
    command set 4
fi
```

```
#!/bin/bash
# nested if
var=4

if [ $var -eq 1 ]
then
    echo "the value is 1"

elif [ $var -eq 2 ]
then
    echo "the value is 2"

elif [ $var -eq 3 ]
then
    echo "the value is 3"

else
    echo "the value is not 1,2,3"
fi
```

```
[wbguo@login1 day3]$ bash nested_if.sh
the value is not 1,2,3
```

Conditions in if-then statement

- Numerical comparison

```
if [ condition ]  
then  
    commands  
fi
```

- String comparison

- File/folder existence

TABLE 12-1 The test Numeric Comparisons

Comparison	Description
<code>n1 -eq n2</code>	Checks if <code>n1</code> is equal to <code>n2</code>
<code>n1 -ge n2</code>	Checks if <code>n1</code> is greater than or equal to <code>n2</code>
<code>n1 -gt n2</code>	Checks if <code>n1</code> is greater than <code>n2</code>
<code>n1 -le n2</code>	Checks if <code>n1</code> is less than or equal to <code>n2</code>
<code>n1 -lt n2</code>	Checks if <code>n1</code> is less than <code>n2</code>
<code>n1 -ne n2</code>	Checks if <code>n1</code> is not equal to <code>n2</code>

TABLE 12-2 The test String Comparisons

Comparison	Description
<code>str1 = str2</code>	Checks if <code>str1</code> is the same as string <code>str2</code>
<code>str1 != str2</code>	Checks if <code>str1</code> is not the same as <code>str2</code>
<code>str1 < str2</code>	Checks if <code>str1</code> is less than <code>str2</code>
<code>str1 > str2</code>	Checks if <code>str1</code> is greater than <code>str2</code>
<code>-n str1</code>	Checks if <code>str1</code> has a length greater than zero
<code>-z str1</code>	Checks if <code>str1</code> has a length of zero

TABLE 12-3 The test File Comparisons

Comparison	Description
<code>-d file</code>	Checks if <code>file</code> exists and is a directory
<code>-e file</code>	Checks if <code>file</code> exists
<code>-f file</code>	Checks if <code>file</code> exists and is a file

Compound condition

- Syntax

- `[condition1] && [condition2]`

- `[condition1] || [condition2]`

```
#/bin/bash
# compound condition
var=3
if [ $var -ge 2 ] && [ $var -le 4 ]
then
    echo "the value is great equal than 2; but less equal than 4"
fi
█
```

```
[wbguo@login1 day3]$ bash compound_condition.sh
the value is great equal than 2; but less equal than 4
```

Practice:



- Create a nested `if-then` script for the following description
 - If tomorrow is `sunny`, then I will `go to the beach`; if it's `rainy`, I will `wash my car`; `otherwise`, I will `stay home`
- Create a script that can check if there is a file called `test_if.txt` in `day3` folder, if it exists, print message to the output. Otherwise, create a empty file called `test`

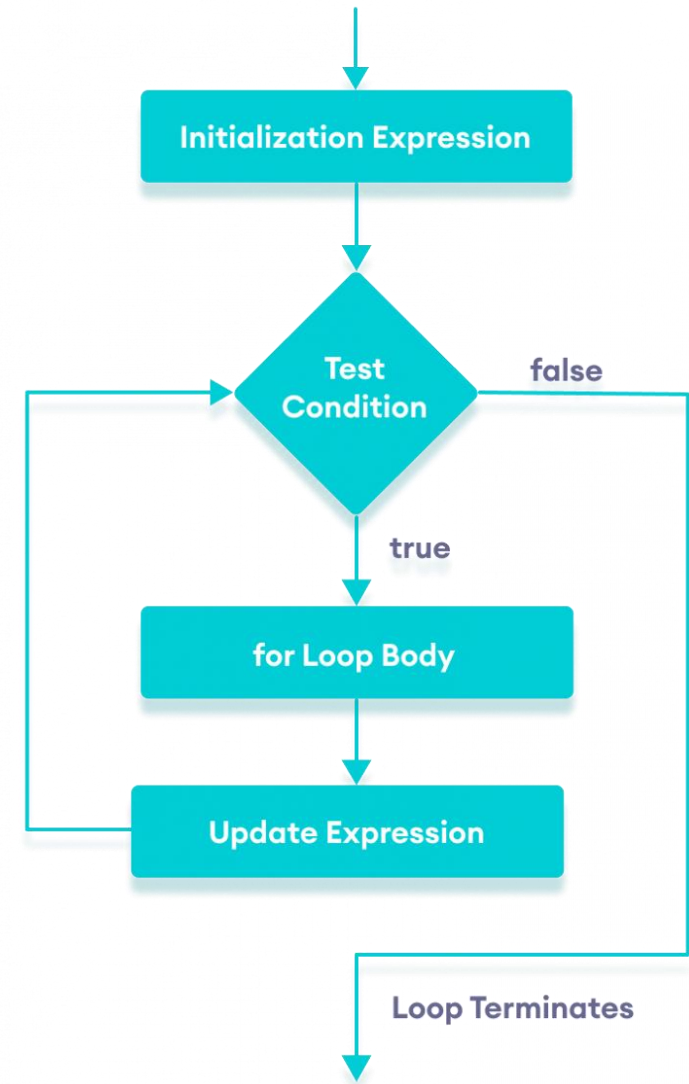
Loop structure

- for loop
- while loop
- Controlling the loop (loop + condition statement)

for loop

- Syntax:

```
for var in list  
do  
    commands  
done
```



for loop

- Syntax:

```
for var in list
do
    commands
done
```

```
#!/bin/bash
# for loop (from variable)
value_list="sample1 sample2 sample3"
for i in $value_list
do
    echo $i
done
```

```
[wbguo@login1 day3]$ bash for_loop.sh
sample1
sample2
sample3
```

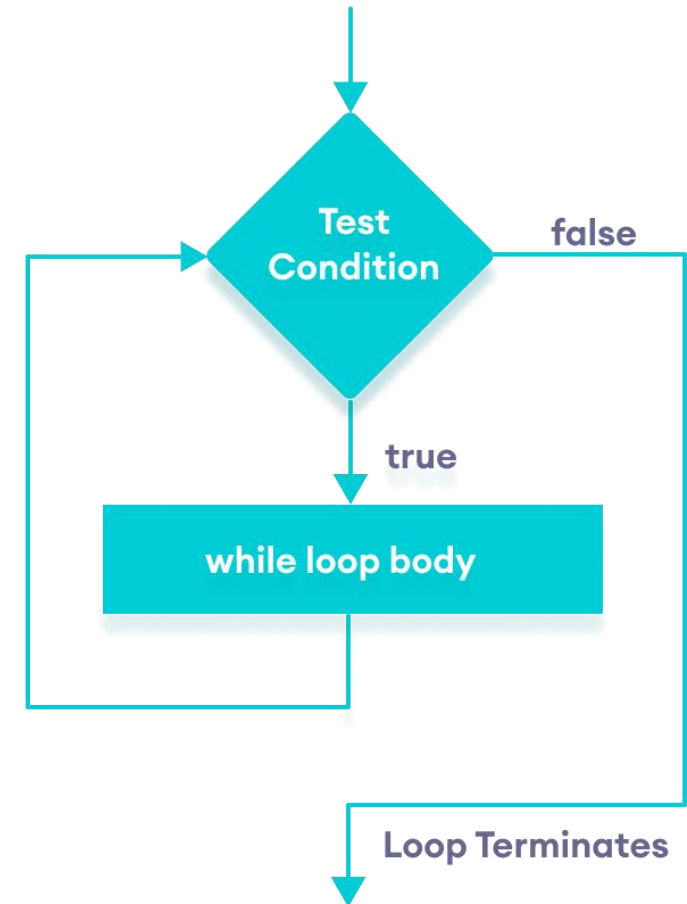
```
#!/bin/bash
# for loop (from array)
value_list=("sample1" "sample2" "sample3")
for i in ${value_list[@]}
do
    echo $i
done
```

```
[wbguo@login1 day3]$ bash for_loop2.sh
sample1
sample2
sample3
```

while loop

- Syntax:

```
while test command → [ condition ]  
do  
    other commands  
done
```



while loop

- Syntax:

```
while test command
do
    other commands
done
```

Note: the **var** must get updated in the loop body Otherwise, the loop will not stop

```
#!/bin/bash

# while loop
var=5

while [ $var -gt 0 ]
do
    echo $var
    var=$(( $var - 1 ))
done
```

```
[wbguo@login1 day3]$ bash while_loop.sh
5
4
3
2
1
```

Controlling the loop

- break command
 - When executed, the break command will **escape the current loop**

```
#!/bin/bash
# loop break
var=5

while [ $var -gt 0 ]
do
    var=$(( $var - 1 ))

    if [ $var -eq 2 ]
    then
        break
    fi

    echo $var
done
```

```
[wbguo@login1 day3]$ bash loop_break.sh
4
3
```


Controlling the loop

- continue command
 - When executed, the continue command will prematurely **stop processing commands** inside of a loop **but not terminate the loop** completely

```
#!/bin/bash

# loop continue
var=5

while [ $var -gt 0 ]
do
    var=$(( $var - 1 ))

    if [ $var -eq 2 ]
    then
        continue
    fi

    echo $var
done
```

```
[wbguo@login1 day3]$ bash loop_continue.sh
4
3
1
0
```

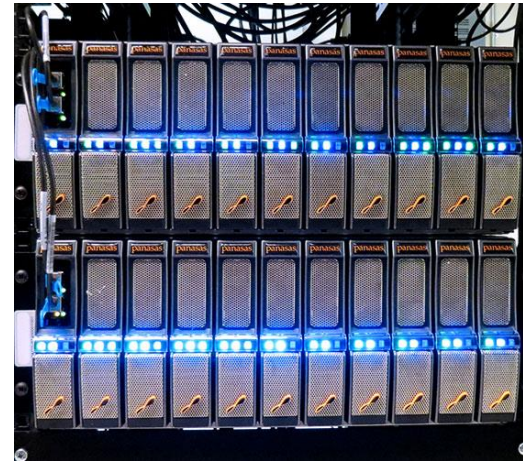
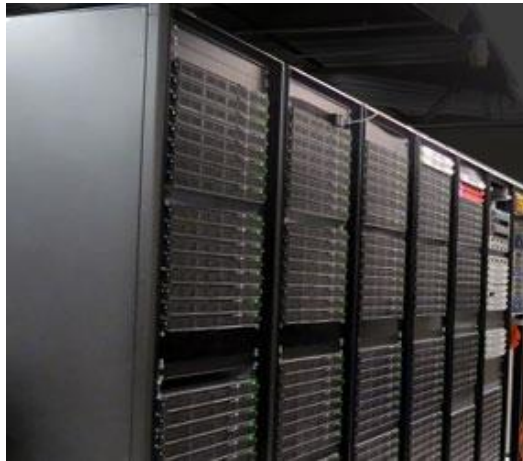
Practice:



- Write a script to read in `test_break_continue.txt` file, and do the following with controlled loop
 - Output lines, if A04 appears then stop
 - Output lines, if A04 appears then skip it

Working with hoffman2

- Useful Commands on hoffman2
- Submit **single job** / **batch jobs** to hoffman2



Useful commands – module

- Check out the list of available tools
 - `module avail`
- Check out the list of loaded tools
 - `module list`
- Load/unload a software to current shell environment
 - `module load/unload <module name>`

Useful commands – my*

- Check out your job status
 - myjobs
- Check out your used/available storage
 - myquota

Useful commands – my*

- Check out your job status
 - `myjobs`
- Check out your used/available storage
 - `myquota`

For users

- Home directory: 40GB (check out the path using `echo $HOME`)
- Project directory: (depends on your group)
- Scratch directory: 2TB (check out the path using `echo $SCRATCH`)

To create a symbolic link of scratch folder in your home directory

- `cd ~`
- `ln -s $SCRATCH tmp`

Useful commands – my*

- Check out your job status
 - myjobs
- Check out your used/available storage
 - myquota

For users

- Home directory: 40GB (check out the path using echo \$HOME)
- Project directory: (depends on your group)
- Scratch directory: 2TB (check out the path using echo \$SCRATCH)

Note: Scratch directory has **no backup**, and files will be erased if they remain unchanged for more than **14 days**

It's a **temporary folder**, users should **not store important files there**

Useful commands – q*

- Request a interactive node
 - `qssh -l h_rt=02:00:00,h_data=1G`
- Submit jobs to the hoffman2 computing cluster
 - `qsub submission_script.sh`
- Delete a submitted job (first use myjobs to find the `job_id`)
 - `qdel job_id`

Submit single job to hoffman2

- code (sjob.sh under folder day3)

```
#!/bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -j y      # Error stream is merged with the standard output
#$ -l h_data=256M,h_rt=00:10:00
#$ -r n      # job is NOT rerunnable
#$ -m a      # Email on abort
#$ -o joblog.$JOB_ID

echo "job $JOB_ID started!"
date

#### put your command here ####
### e.g. python myscript.py ###
sleep 300
#### put your command here ####

echo "job $JOB_ID Finished!"
date
```

Submit single job to hoffman2

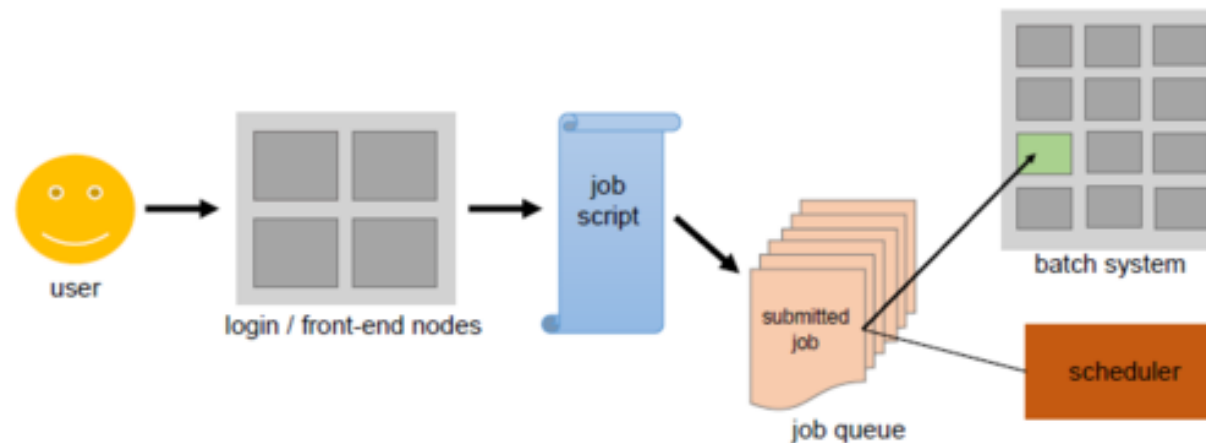
- Submit with qsub

```
[wbguo@login1 day3]$ ls
example_sh jobs qjob.sh sjob.sh test_break_continue.txt
[wbguo@login1 day3]$ qsub sjob.sh
Your job 99765 ("sjob.sh") has been submitted
[wbguo@login1 day3]$ myjobs
job-ID      prior    name          user      state submit/start at   queue                jclass                               slots ja-task-ID
-----
    99765    0.00000 sjob.sh       wbguo      qw    09/30/2021 07:08:42
[wbguo@login1 day3]$ myjobs
job-ID      prior    name          user      state submit/start at   queue                jclass                               slots ja-task-ID
-----
    99765    0.50500 sjob.sh       wbguo      r    09/30/2021 07:08:49 msa_smp.q@n2236
[wbguo@login1 day3]$ ls
example_sh joblog.99765 jobs qjob.sh sjob.sh test_break_continue.txt
[wbguo@login1 day3]$
```

```
[wbguo@login1 day3]$ ls
example_sh joblog.99765 jobs qjob.sh sjob.sh test_break_continue.txt
[wbguo@login1 day3]$ cat joblog.99765
job 99765 started!
Thu Sep 30 07:08:49 PDT 2021
job 99765 Finished!
Thu Sep 30 07:13:49 PDT 2021
```

Submit batch jobs to hoffman2

- Assume you have 100 samples
 - Sequentially processing with a for loop (total time: $100 * t$)
 - Parallely processing by the job scheduler (total time: $100/n * t$)



- t is the time needed for processing a single sample
- n is the #jobs that scheduled to run simultaneously

Submit batch jobs to hoffman2

- code (qjob.sh under folder day3)

```
#!/bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -t 1-5    # because we have 5 samples
#$ -j y      # Error stream is merged with the standard output
#$ -l h_data=256M,h_rt=00:10:00
#$ -r n      # job is NOT rerunnable
#$ -m a      # Email on abort
#$ -o joblog.$JOB_ID.$TASK_ID

ja=$HOME/qcbio-intro2Unix/day3/jobs

PARMS=( $(awk "NR==$SGE_TASK_ID" $ja) )
sample_name=${PARMS[0]}

echo "$sample_name started!"
date

##### put your command here #####
### e.g. STAR ${sample_name}.fastq.gz -o ${sample_name}.bam ###
sleep 300
##### put your command here #####

echo "$sample_name finished!"
date
```

Submit batch jobs to hoffman2

- Submit with qsub

```
[wbguo@login1 day3]$ ls
example_sh jobs qjob.sh sjob.sh test_break_continue.txt
[wbguo@login1 day3]$ qsub qjob.sh
Your job-array 99807.1-5:1 ("qjob.sh") has been submitted
[wbguo@login1 day3]$ myjobs
```

job-ID	prior	name	user	state	submit/start at	queue	jclass	slots	ja-task-ID
99807	0.50500	qjob.sh	wbguo	r	09/30/2021 07:21:39	msa_smp.q@n2191		1	1
99807	0.50500	qjob.sh	wbguo	r	09/30/2021 07:21:39	msa_smp.q@n2198		1	2
99807	0.50500	qjob.sh	wbguo	r	09/30/2021 07:21:39	pod_ib100.q@n7645		1	3
99807	0.50500	qjob.sh	wbguo	r	09/30/2021 07:21:39	pod_ib100.q@n7645		1	4
99807	0.50500	qjob.sh	wbguo	r	09/30/2021 07:21:39	pod_ib56_xgold.q@n6367		1	5

```
[wbguo@login1 day3]$ ls
example_sh joblog.99807.1 joblog.99807.2 joblog.99807.3 joblog.99807.4 joblog.99807.5 jobs qjob.sh sjob.sh test_break_continue.txt
[wbguo@login1 day3]$
```

```
[wbguo@login1 day3]$ ls
example_sh joblog.99807.1 joblog.99807.2 joblog.99807.3 joblog.99807.4 joblog.99807.5 jobs qjob.sh sjob.sh test_break_continue.txt
[wbguo@login1 day3]$ cat joblog.99807.1
sample1 started!
Thu Sep 30 07:21:39 PDT 2021
sample1 finished!
Thu Sep 30 07:26:39 PDT 2021
[wbguo@login1 day3]$
```

Tips for how much resources to request

- Request an **interactive node** with **relatively large** resources
- Run a single job as a **test run** on the interactive node
- Use command `top` to check the memory consumption
- Estimate the finished time (e.g. using `date` when the job is finished)
- Then the `h_rt, h_data` is all clear!

Practice



- Load `samtools` into your shell environment
- Submit `sjobs.sh` and `qjobs.sh` to `hoffman2`
- After submission, perform `myjobs`, `qdel` to delete jobs

Common errors and solutions

- No such file or directory
 - Check the file using ls
 - Check the path

```
[wbguo@login1 day1]$ cat abcd.txt
cat: abcd.txt: No such file or directory
[wbguo@login1 day1]$ ls
test
[wbguo@login1 day1]$ █
```


Common errors and solutions

- Command not found
 - Check the command (if there is a typo?)
 - Check if the software is loaded

```
[wbguo@login1 day1]$ LS
bash: LS: command not found
[wbguo@login1 day1]$ ls
test
```

```
[wbguo@login1 day1]$ conda
bash: conda: command not found
[wbguo@login1 day1]$ module load anaconda3
[wbguo@login1 day1]$ conda --help
usage: conda [-h] [-V] command ...

conda is a tool for managing and deploying applications, environments and packages.

Options:
positional arguments:
  command
  clean                Remove unused packages and caches.
  compare              Compare packages between conda environments.
  config               Modify configuration values in .condarc. This is modeled after the git config command. Writes to the
                        user .condarc file (/u/home/w/wbguo/.condarc) by default.
  create               Create a new conda environment from a list of specified packages.
  help                 Displays a list of available conda commands and their help strings.
  info                 Display information about current conda install.
  init                 Initialize conda for shell interaction. [Experimental]
  install              Installs a list of packages into a specified conda environment.
  list                 List linked packages in a conda environment.
  package              Low-level conda package utility. (EXPERIMENTAL)
  remove               Remove a list of packages from a specified conda environment.
  uninstall            Alias for conda remove.
  run                  Run an executable in a conda environment. [Experimental]
  search               Search for packages and display associated information. The input is a MatchSpec, a query language for
                        conda packages. See examples below.
  update               Updates conda packages to the latest compatible version.
  upgrade              Alias for conda update.

optional arguments:
  -h, --help            Show this help message and exit.
```

Summary

We have knowledge about

- Unix file system and how to navigate through it
- Manipulating the files and directories
- Advanced Command lines for file processing
- sed, awk, grep for contents editing/extracting/searching
- Regular expression (RegEx)
- Shell programming (variables' operation/ if condition/ loop structure...)

And we have experience in

- Accessing hoffman2 cluster & file transfer
- Requesting resources on hoffman2
- Executing commands and using pipe
- Submitting and managing jobs on hoffman2



You are almost an Linux expert!

Summary

- command

Command	Function
echo	Print string to standard output
date	Show the date/time
myquota	Show storage quota
myjobs	Show jobs status
module load	Load an tool into environment
module avail	Show the available tools
module list	Show tools that has been loaded
qrsh	Request a interactive node
qsub	Submit a job/jobs
qdel	Delete a job

Q&A

[Google doc](#)

Where to get help?

- <https://www.google.com>
- <https://stackoverflow.com>
- <https://unix.stackexchange.com>
- https://www.tutorialspoint.com/unix/shell_scripting.htm



Lastly...GLHF!

```
-----  
< Good Luck, Have Fun! >  
-----  
      ^  ^  
      --  
      (oo)\  
      ( _ )\  )\/\
```

And **don't compute** on the **login node**!

Thank you!