Shell Command & Shell Script

2023-2024

COMP2113B/C Programming Technologies / ENGG1340B/C Computer Programming II

Dr. Chenxiong Qian & Dr. T.W. Chim (E-mail: cqian@cs.hku.hk & twchim@cs.hku.hk)

Department of Computer Science, The University of Hong Kong

We are going to learn…

- Useful Shell commands
- Shell Script
  - A “Hello World” example
  - Variables
  - No quote, ‘single quote’ and “double quote”
  - String operations
  - Flow of control (if-else, for loop)
  - Mathematical operations
  - Arguments

What is the programming syntax of shell script?

Useful Shell Commands

- *Please refer to Module 1 for details!*
- *You must try them in order to remember them!*

Directory manipulations

Command	Meaning
pwd	n fb present working directory…
ls ls -l	… fb n ; fb ; ; n fb ; …
cd dir cd ~ cd .. cd .	changes dir… n … … … ; n n w ffb …
mkdir dir	creates n dir…
rmdir dir	removes dir… fbdir empty…
rm -rf dir	removes the non empty directory dir c fb ..
mv dir dir2	fbdir2 does not exist; renames fb n dir dir2… ; moves dir dir2…
cp –r dir1 dir2	copy dir1 dir2 c

File manipulations 1

Command	Meaning
pico a.cpp	n y fb a.cpp…) ” ” w ffbb n n [vi vim)W N w [N …
g++ a.cpp -o a.o	w n n n a.cpp y c a.o… y c n w y …y …
./a.o	w n a.o…

File manipulations 2

Command	Meaning
cp file1 file2	Copy file1 file2…
mv file dir mv file1 file2 mv dir1 dir2	fb dir ; moves file dir… fb n n) ….;.c file1 file2 fb [; renames file1 file2… n fb … fb dir2 y ; mv n w dir1 dir2…
rm file rm –rf dir	Remove file… Remove w fb dir.
touch file	Create an empty file n file…
cat file	Display the content fb file…

Others

Command	Meaning
wc file	counts n c fb ; ; file...
sort file	sorts fbfile c ...
cut -d, -f1 file	fb columns of data... w n fb c fb –d; n fb c fb –f) fb n c fb n [...
grep 'abc' file	returns the lines fb abc"... N n c fb y) fb -E[...
uniq file	removes adjacent duplicate fb n ...
diff file1 file2	different file1 file2... w ; diff n n n c fb n ...
spell file	incorrect words file...

Examples

- What is the full path of your default directory when you startup your shell?

```
/    n  /          /       n
```

present working directory

c ffb ...

- What are the directories in the root directory?

```
      /

      fb
```

- How to go back to your home directory?

```

```

Examples

- **Copy** the source code hello.cpp **to** hello2.cpp

```
            …              …
```

- **Rename** hello2.cpp **to** backup.cpp

```
 n w        …     c        …
```

- **Create** a directory "backup" **and move** backup.cpp **in it.**

```
 n        c
 n wc          …     c
```

Wildcards

- The Linux shell has a mechanism to generate a list of file names matching a pattern

Wildcard	Meaning
	N string …
	N character…

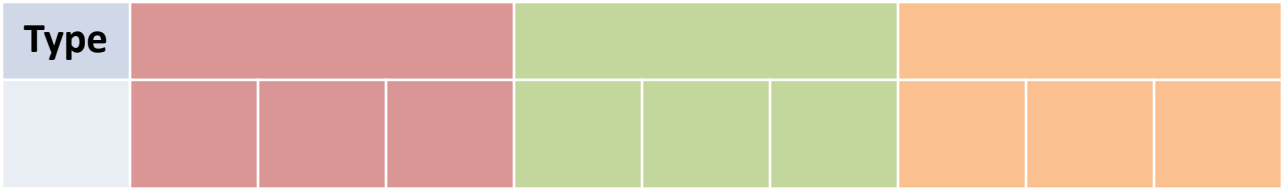
```
n w … c
c
…
hello.cpp hello.o
```

File permission & security

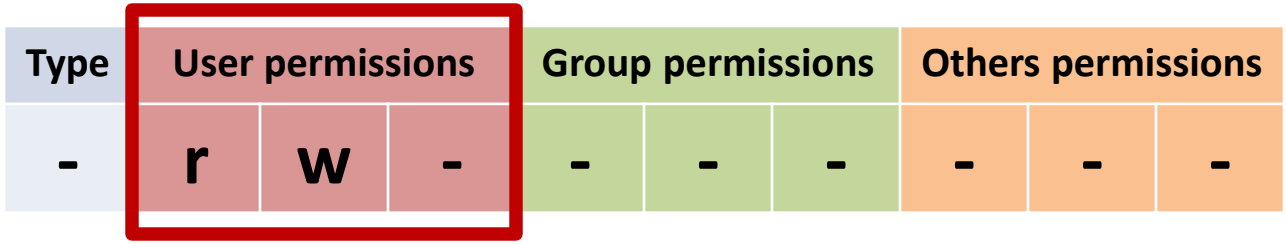
- You can use the list directory command ls -l to return the permission code of files / directories.

```
        fb
      fb
-rw--------…          n                          .     fb
```

File permission & security



File permission & security



- **User permissions**
  - c **Read)r[; Write)w[; Execute)x[** n
    fb fb fb …
  - C n ”**rw-**”; **R**
    W fb ; c **x** fb …

File permission & security

- n fb fb /

chmod]who{operator]permissions{ fb n

who

value	meaning
u	) [
g	
o	
a	) ; [

operator

value	meaning
+	n
-	n w n
=	n

permissions

value	meaning
r	n
w	n
x	y n

-)+[y)x[n)u[.
-)+[ )r[ )w[n)a[.
- n w)-[ )r[ )w[fb n)g[ )o[

n y fb

n fb

n fb

Examples

- List the permission of the files with prefix “hello.”

```
…
…  n  .  …
…  n  .  …
```

- Take away the execute permission (x) on hello.o from user (u), what will happen?

```
n  y  …
./  …
c  . ./  … .  n
```

Shell Script

Please refer to Module 2 for details!

Motivation

fb

n n N ...

y r fb

n n c

n n w

n ;

```
#!/bin/bash
g++ ... -o ...
./. ... < ...y > ...y
sort ...y | uniq > r ...y
spell r ...y > n ...y
diff r ...y n ...y | grep -E
...
```

```
./. ...
< loop
< polo
< pool
```

Answer: c w

n n fb ...

shell script...

A Hello World Example

My first shell script

- #!/bin/bash n fb fb

  …

- fb #! n c

  … ;

  bash n) bash shell[…

#!/bin/bash

…

#!/bin/bash

- ffb n n Bash

  shell ffb …

  which bash n n

  Bash shell…

Comments and echo

```
#!/bin/bash
# This is a comment
echo "Hello world!"
...
```

Commenting

- y fb ; fb # comment
  ...

echo "Hello world!"

- echo n n
  fb w
  fb w c ...
- fb -n fb
  ...

```
echo -n "Hello World!"
```

Execute(x) permission to run

- c n y c c **granting**
)+[**execute**)x[n **user**)u[…

No execute (x) permission!

- w y
  n y

```
#!/bin/bash
# This is a comment
echo "Hello world!"
```

…

./ …
C . ./ … . n

n y …
./ …

Shell script is very useful

- n n fb n c fb
  n n c ...

Note: < and > are used to redirect input from a file and redirect output to a file.

```
//add.cpp
#include   iostream
using namespace std:
int main()}
  int a; b; c:
  cin     a     b     c:
  cout     a    b    c          :
(
```

...

3 4 5

...y

12

...y

```
#!/bin/bash
      n
g++ add.cpp     add.o

./.add.o    input.txt    output.txt

cat output.txt
```

y n ...

VERY USEFUL ☺!

n ;

y c ;

J c ./hello.sh

n y y n ...

./. y n ...

An interpreted language

- **interpreted language**; c
  n …
  - n **parsed and interpreted by the shell every time the program is executed**…
  - ; n c
    y c fb n c fb y n …
- **modify the program more quickly** c n …
  - w ; n **slower** c
    y n …

Variables

string variable only

- w c . string...
- W c n case sensitive...
- fb n w c *a*
  w cat ...

a cat

No space! No space!

IMPORTANT!!!!!
There must be NO SPACE before and after the = sign.

- $ retrieve the value fb w c ...

echo *a*

A space

Spacing is critical!

Space before and after =

- BEFORE AFTER will cause problem

w c command...

```
#!/bin/bash
a
echo a
```

y n ...

```
./ y n ...
./ y n ... . . a: command not found
```

NO space before and after =

- [Setting value] NO $ sign
  setting w fb w c ...
- [Retrieve value] Use $ sign
  retrieving w fb w c ...

```
#!/bin/bash
a
echo a
```

y n c...

```
./ y n c...
```

Variables

- w c **without declaration…**
- C **creates the variable automatically when a variable is used…**

The read command

- read n n
  fb n
  w c fb n n …
- w c ***name***
  n ;
  …

```
#!/bin/bash
echo                    n
read name
```

y n …

```
./. y n …
                n
Chim
```

Variables

- w c **without declaration…**
- C **creates the variable automatically when a variable is used…**

Use $ when retrieve value

- **[Retrieving value]**
  r **retrieving** w
  fb w c …

```
#!/bin/bash
echo                    n
read name
echo "          name"
```

y n …

```
./. y n       …
                  n
   n
Hello Chim
```

Quoting

Specifying strings

- **Quoting** w n n n
  ...
  - Unquoted
  - 'Single quote'
  - "Double quote"

Unquoted

- fb w **without any quoting;**
  c n fb w
  fb **single word**…

Error: Unquoted word with space

- ; ”pie”
  command; fb
  command not found…

```
#!/bin/bash
a
echo a
b
echo b
```

y n …

./ y n …

./ y n … . . **pie: command not found**

Single quote

- w c fb single quote c
  w fb ...

```
#!/bin/bash
a
echo a
```

y n ...

```
./. y n ...
```

Single quote

- w c fb**single quote** c
  w fb ...
  - w ; **does not support variable substitution**...

```
#!/bin/bash
a
echo a
b \
echo b
```

y n ...

\ ;

NOT

c c
w fbw c ***a*** fb
single quote...

./. y n ...

$a\$

Double quote

- ffb fb n r ; **double quote will handle three special characters** fb n …

Symbol	Meaning
	Dollar sign - W c c …
\	**Backslash -** …
	Back quotes - c n n

c

Where is the back quote button on keyboard?

c r

Double quote

```
#!/bin/bash
a
b a
echo b
```

y n ...

```
./. y n ...
Apple pie
```

Supports value with space

- c r n w
  ...

Supports variable substitution

- c r variable substitution;
  fb w fbw c *b* Apple pie
  c NOT $a ...

Double quote

```
#!/bin/bash
a
b    a
echo  b
c  \        a
echo  c
d   ls
echo  d
```

y n ...

```
./. y n        ...

$a = Apple pie
example6.sh
```

Supports escape characters

- escape characters
  c r ...
- ....;.\$
  \$a c c
  w fbw c *a*

`Back quote` = shell command!

- C r n shell command...
- fb ; `ls` c y
  command c the result of the
  command...
) ....;.`ls` fb
  ; example6.sh
  y n [

Double quote

- **back quotes** ; fb
  command w c fb fb ...

c

fb

```
#!/bin/bash
a          fb
echo  a
```

y n ...

```
./. y n        ...
Apple Banana Cherry
```

Question:

c

each

word fb

Answer:

n "**for** "

n n ;

c .

String operations

String Operation

- fb n n c fb ...
  -
  - c
  - fb

String length

- **String length.** w *a*; fb
  n c fb *a*...

${#a}

```
#!/bin/bash
a
echo fb\ $a\ ${#a}
```

y n ...

```
./. y n ...
fb' 5
```

\" is an escape character

${#a} returns the string length (i.e., 5)

Substring

- **Substring (use ":").** w *a*; fb c fb*a* fb n *pos* *len*..

${*a*:*pos*:*len*}

```
#!/bin/bash
a
echo $a c . ${a:5:5}
```

y n ...

./. y n ...
c . apple

P	i	n	e		a	p	p	l	e

Note: fb
w y ...

Replace

- **Replace)** ”/”[... w *a*; fb
  fb n c **first**
  occurrence fb ***from*** ***to***...

${*a*/*from*/*to*}

```
#!/bin/bash
a
from
to  J
echo -n                fb          \ $from\  c  \ $to\
echo  c     n   ${a/$from/$to}
```

```
./.  y  n        ...
            fb              c  J     c    n   Apple juice
```

Note: fb -n;
echo w
y **echo**

y n ...

n ...

Flow of Control

If-else statement

- c y fb **if** n c …

```
if [ condition ]
then
        fb  n      n
fi
```

```
if [ condition 1 ]
then
   echo
elif [ condition 2 ]
then
   echo
else
   echo
fi
```

[condition] for string

fb n n ...

String comparisons	Meaning
[*string*]	True ffb fb *string*
[*string1* == *string2*]	True ffb r
[*string1* != *string2*]	True ffb ffb
[*string1* \> *string2*]	True ffb *string1* fb *string2*
[*string1* \< *string2*]	True ffb *string1* c fb *string2*

string1 *string2*

double quote n **work even if there are spaces inside** *string1* *string2*...

Spacing is critical!

```
if ["$ans"=="Y"]
```

- fb NO SPACE c
  n ;
  n
  n command...
  fb "Command not found"

```
./. y n ...
n w ... fb ) / [

./example11.sh: [Y==Y] command not found
```

```
#!/bin/bash
echo n w ... fb ) / [
read ans
if [ ans ]
then
    rm -rf *.cpp
    echo ... fb n w
fi
y n ... ) [
```

```
if [ "$ans" == "Y" ]
```

space space space space space

[condition] for file

- w w fb
  ...

File checking	Meaning
[-e *file*]	True fb*file* exists...
[-f *file*]	True fb*file* is a file...
[-d *file*]	True fb*file* is a directory...
[-s *file*]	True fb*file* has size > 0...
[-r *file*]	True fb*file* is readable...
[-w *file*]	True fb*file* is writable...
[-x *file*]	True fb*file* is executable...

Note: fb n
y ...

[condition] for file

- fb hello.cpp
  y .

```
./ y n ...
hello.cpp not found!
```

n n
compile **run**
hello.cpp;
n) [fb ...

```
#!/bin/bash
if [ -e hello.cpp ]
then

else
    echo ... fb
fi
```

y n ...

[condition] for file

- fb hello.cpp fb

```
./.  y  n          ...
Hello World!
```

```
#!/bin/bash
if  [ -e hello.cpp ]
then
    rm *.o
    g++ hello.cpp -o hello.o
    if [ -e hello.o ]
    then
        ./hello.o



    fi
else
    echo        ...        fb
fi
                                y  n        ...
```

[condition] for file

- fb hello.cpp
  n .

```
./. y n ...
Compilation failed!
) n
c n [
```

```
#!/bin/bash
if [ -e hello.cpp ]
then
    rm *.o
    g++ hello.cpp -o hello.o 2> error.txt
    if [ -e hello.o ]
    then
        ./hello.o
    else
        echo n fb
        cat error.txt
    fi
else
    echo ... fb
fi
```

y n ...

[condition] for command

- c **shell command**...
- w fb n n

y fb ...

```
./. y n ...
cp: cannot stat `file'123 : No such file or directory
Command failed
fb
./. y n ...
Command executed successfully
fb
file123 fileabc
```

```
#!/bin/bash
if cp file123 fileabc
then
  echo n n y fb
else
  echo n n fb
fi
```

y n ...

for loop

- for fb ...

```
#!/bin/bash
list
for i in $list
do
  echo i
done
```

y n ...

```
./. y n ...
1
2
3
4
5
```

for loop

```
#!/bin/bash
list `ls *.cpp`
for name in list
do
  cp name name.c
done
```

c …

`ls *.cpp`

n n ls *.cpp

fb .cpp

fb fby)fb y […

… c…

./c …

…

… a.cpp.backup

c… b.cpp.backup

for ;

c .cpp fb

☺

Mathematics operations

Mathematics operations

- n n fb n n
  … w ; fb n
  n n **let** n n …

```
#!/bin/bash
a 10
let a a a a/ a
echo a
```

y n …

```
./. y n …
```

[condition] for numbers

- fb fb n n n n
  fb if n ;
  y
  …

Integer comparisons	Meaning
[*a* -eq *b*]	True fl*a* *b*
[*a* -ne *b*]	True ffl*a* *b*
[*a* -lt *b*]	True ffl*a* *b*
[*a* -le *b*]	True ffl*a* *b*
[*a* -gt *b*]	True ffl*a* *b*
[*a* -ge *b*]	True ffl*a* *b*

Mathematics operations

- n

  ffb ...

```
./. y n ...
C n ;
C n ;
```

```
#!/bin/bash
a 100
b 99
echo C n ;
if [ a \> b ]
then
 echo
else
 echo c
fi
```

```
echo C n ;
if [ a -gt b ]
then
 echo
else
 echo c
fi
```

y n ...

Arguments

Getting arguments

- **Command line arguments c *$0*; *$1*; .....**
  $*9*... ; *$0* n fb ...
  - n n n fb ***$9*** c **${*10*}**; .....
  - **$#** n c fb n y
    ...

```
./. y n ... n
        3 n
    ./example18.sh
    sun
    mon
    tue
```

```
#!/bin/bash
echo "There are $# arguments"
echo \ $0
echo \ $1
echo \ $2
echo \ $3
```

y n ...

END

2023-2024

COMP2113B/C Programming Technologies / ENGG1340B/C Computer Programming II

Dr. Chenxiong Qian & Dr. T.W. Chim (E-mail: cqian@cs.hku.hk & twchim@cs.hku.hk)

Department of Computer Science, The University of Hong Kong