

# Introduction to Shell programming

Tutor: Lưu Thanh Trà  
Email: [lttra@hoasen.edu.vn](mailto:lttra@hoasen.edu.vn)

# Outline

---

- Introduction
- Common commands
- Control structure

# Shell

---

- A program whose job was executing other programs on behalf of users.
- A variety of shell: C shell (csh), Bourne shell (sh), Bash shell (bash),...
- Quick, effective tool that remembers typed commands, allow to define your own command abbreviations, shortcuts, and other features.
- Shells are programmable

# Commands

---

- Where are you?
  - pwd
    - Usage: locate the current path
- Running commands
  - type
    - Usage: show the path of a command
    - Ex: type ls
  - type -a
    - Usage: show the alias of a command alias
    - Ex:
      - alias lf="ls -la"
      - type -a lf

## ■ which

- Ex: which ls

## ■ apropos:

- Usage: search manpage names and description for regular expression
- Ex: apropos hash
- Equal to : man -k hash

## ■ locate/slocate

- Usage: consult database files about the system
- Ex: locate apropos
- Note: update the dbase with updatedb to update the file /var/lib/mlocate/mlocate.db

# File information

---

## ■ ls

- Usage: list files in a directory
- Ex: `ls -la /tmps`
- Ex: `ls -ld /tmps/*`

## ■ stat

- Usage: display file/filesystem status
- Ex: `stat /root/Desktop/file1`

## ■ file:

- Usage: determine file type
- Ex: `file /root/Desktop/pic1.jpg`

# ls command

---

- **-a**
  - Do not hide files starting with . (dot)
- **-F**
  - Show the type of file with one of these trailing type designators: /\*@%|=|
- **-l**
  - Long listing
- **-L**
  - Show information about the linked file, rather than the symbolic link itself
- **-Q**
  - Quote names (GNU extension, not supported on all systems)

---

- *-r*

- Reverse sort order

- *-R*

- Recurse though subdirectories

- *-S*

- Sort by file size

- *-l*


- Short format but only one file per line

---



# Shell quoting

---

- `$ echo A coffee is $5?!`
  - *A coffee is ?!* 
- `$ echo "A coffee is $5?!"`
  - *-bash: !": event not found*
- `$ echo 'A coffee is $5?!'`
  - *A coffee is \$5?!*
- `$ echo 'A coffee is $5 for' "$USER" '?!'`
  - *A coffee is \$5 for jp ?!*
- `$ echo "A coffee is \$5 for $USER?\\!"`
  - *A coffee is \$5 for jp?\\!*

# Built-in and external commands

---

- Built-in commands are integrated in the shell
- External commands can be found on in \$PATH
- enable:
  - Usage: stop using built-in command
  - enable -n cd
  - builtin cd

# Interactive mode

---

- In the interactive mode, systems ask users to confirm commands

```
#!/usr/bin/env bash
# cookbook filename: interactive
case "$-" in
*i*) # Code for interactive shell here
;;
*) # Code for non-interactive shell here
;;
esac
```

- Note: \$- is string listing of all the current shell option flags

# Setting bash as default shell

---

■ ?

# Writing a function

---

■ **function** *function\_name*

```
{  
  commands to execute  
}
```

or

■ *function\_name* ()

```
{  
  commands to execute  
}
```

# Declare the shell

---

- `#!/usr/bin/sh` **OR** `#!/bin/sh`
- `#!/usr/bin/ksh` **OR** `#!/bin/ksh`
- `#!/usr/bin/csh` **OR** `#!/bin/csh`
- `#!/usr/bin/bash` **OR** `#!/bin/bash`

# Control structures

---

**if ... then Statement**

if [ **test\_command** ]

then

commands

fi

**if ... then ... else Statement**

if [ **test\_command** ]

then

commands

else

commands

fi

# for ...in...

---

```
for loop_variable in argument_list  
do  
  commands  
done
```



# while

---

```
while test_command_is_true  
do  
  commands  
done
```

# until

---

```
until test_command_is_true  
do  
  commands  
done
```

# case

---

- `case $variable in`
- `match_1)`
- `commands_to_execute_for_1`
- `::`
- `match_n)`
- `commands_to_execute_for_n`
- `::`
- `*) (Optional - any other value)`
- `commands_to_execute_for_no_match`
- `::`
- `esac`

# break, continue, exit, return,

---

- break
- continue:
- exit
- return:

# Variable

---

- Variable

- ex: VAR=1

- echo \$VAR

- Command-line arguments

- \$1, \$2,....

