Executing commands

Tutor: Lưu Thanh Trà
Email: lttra@hoasen.edu.vn

Outline

- Running a command
- Shift command
- Math operations
- File permissions

Running a command

- To run a command
  - # a_command parameters
- PATH variable
  - PATH=$PATH:/mydir
- Example

```
$ touch ls
$ chmod 755 ls
$ PATH=".:$PATH"
$ ls
```

Returned value

- $?

```
$ somecommand
it works...
$ echo $?
0
$ badcommand
it fails...
$ echo $?
1
```

- $ if (($?)) ; then echo failed ; else echo OK; fi

Running several commands

- Running commands one by one
  - $ long ; medium ; short
- Running commands only if they are successful
  - $ long && medium && short
- Commands in background
  - Example: *long&*
  - Background: bg
  - Foreground: fg
  - jobs
  - kill %1

Deciding whether a command succeeds

- Example

```
cd mytmp
if (( $? )); then rm * ; fi
```

- Or

```
cd mytmp && rm *
```

Long jobs unattended

- nohup

  $ nohup long &

  nohup: appending output to `nohup.out'

Display error messages when failures occur

- cmd || printf "%b" "cmd failed. You're on your own\n"
- && : AND
- || : OR
- cmd || printf "%b" "FAILED.\n" ; exit 1
- cmd || { printf "%b" "FAILED.\n" ; exit 1 ; }

Running commands from a variable

- Example

```
FN=/tmp/x.x
PROG=echo
$PROG $FN
PROG=cat
$PROG $FN
```

- Example 2:

```
for SCRIPT in /path/to/scripts/dir/*
do
if [ -f $SCRIPT -a -x $SCRIPT ]
then
$SCRIPT
fi
done
```

Symbol commands

- ( ) Run the enclosed command in a sub-shell
- (( )) ~~Evaluate and assign value to variable and do math in a shell~~
- $(( )) Evaluate the enclosed expression
- [ ] Same as the test command
- [[ ]] Used for string comparison
- $( ) Command substitution
- `command` Command substitution

Shift commands

```
#!/bin/bash
COUNT=0          # Initialize the counter to zero
NUMBER=$#        # Total number of command-line arguments to process
# Start a while loop
while [ $COUNT -lt $NUMBER ]
do
COUNT=`expr $COUNT + 1`          # A little math in the shell script

# Loops through each token starting with $1 process each $TOKEN
TOKEN='$'$COUNT
shift            # Grab the next token, i.e. $2 becomes $1
done
```

$* vs $@

- $* :
  - special parameter takes the entire list as one argument with spaces between.
- $@
  - special parameter takes the entire list and separates it into separate arguments.
- Example

```
for TOKEN in $*
do
process each $TOKEN
done
```

Math operation

- ++ — Auto-increment and auto-decrement, both prefix and postfix
- + Unary plus
- - Unary minus
- ! ~ Logical negation; binary inversion (one’s complement)
- * / % Multiplication; division; modulus (remainder)
- + - Addition; subtraction
- << >> Bitwise left shift; bitwise right shift
- <= >= Less than or equal to; greater than or equal to

- < > Less than; greater than
- == != Equality; inequality (both evaluated left to right)
- & Bitwise AND
- ^ Bitwise exclusive OR
- | Bitwise OR
- && Logical AND
- || Logical OR

- abs Absolute value
- log Natural logarithm
- acos Arc cosine
- sin Sine
- asin Arc sine
- sinh Hyperbolic sine
- cos Cosine

- sqrt Square root
- cosh Hyperbolic cosine
- tan Tangent
- exp Exponential function
- tanh Hyperbolic tangent
- int Integer part of floating-point number

chmod command

- 4000 Sets user ID on execution.
- 2000 Sets group ID on execution.
- 1000 Sets the link permission to directories or sets the save-text attribute for files.

- 0400 Permits read by owner.
- 0200 Permits write by owner.
- 0100 Permits execute or search by owner.
- 0040 Permits read by group.
- 0020 Permits write by group.
- 0010 Permits execute or search by group.
- 0004 Permits read by others.
- 0002 Permits write by others.
- 0001 Permits execute or search by others.

Traps

- Sending a signal
  - kill –1 *process_id*
- Setting traps
  - trap ‘echo “\nEXITING on a TRAPPED SIGNAL”;exit’ 1 2 3 15

Exit signals

- **0 — Normal termination, end of script**
- **1 SIGHUP Hang up, line disconnected**
- **2 SIGINT Terminal interrupt, usually CONTROL-C**
- **3 SIGQUIT Quit key, child processes to die before terminating**
- **9 SIGKILL kill -9 command, cannot trap this type of exit status**
- **15 SIGTERM kill command's default action**
- **24 SIGSTOP Stop, usually CONTROL-z**

Test condition

-b file = True if the file exists and is block special file.

-c file = True if the file exists and is character special file.

-d file = True if the file exists and is a directory.

-e file = True if the file exists.

-f file = True if the file exists and is a regular file

-g file = True if the file exists and the set-group-id bit is set.

-k file = True if the files' "sticky" bit is set.

-L file = True if the file exists and is a symbolic link.

-p file = True if the file exists and is a named pipe.

-r file = True if the file exists and is readable.

-s file = True if the file exists and its size is greater than zero.

- # -s file = True if the file exists and is a socket.
- # -t fd = True if the file descriptor is opened on a terminal.
- # -u file = True if the file exists and its set-user-id bit is set.
- # -w file = True if the file exists and is writable.
- **# -x file = True if the file exists and is executable.**
- # -O file = True if the file exists and is owned by the effective user id.
- # -G file = True if the file exists and is owned by the effective group id.
- # file1 –nt file2 = True if file1 is newer, by modification date, than file2.
- # file1 ot file2 = True if file1 is older than file2.

- # file1 ef file2 = True if file1 and file2 have the same device and inode numbers.
- **# -z string = True if the length of the string is 0.**
- **# -n string = True if the length of the string is non-zero.**
- **# string1 = string2 = True if the strings are equal.**
- # string1 != string2 = True if the strings are not equal.
- # !expr = True if the expr evaluates to false.
- **# expr1 –a expr2 = True if both expr1 and expr2 are true.**
- **# expr1 –o expr2 = True is either expr1 or expr2 is true.**

Choose function

- # Called like: choice <promtp>
- # e.g. choice "Do you want to play a game?"
- # Returns: global variable CHOICE

```
function choice {
CHOICE=''
local prompt="$*"
local answer
read -p "$prompt" answer
case "$answer" in
[yY1] ) CHOICE='y';;
[nN0] ) CHOICE='n';;
* ) CHOICE="$answer";;
esac
} # end of function choice
```

```
1.  until [ "$CHOICE" = "y" ]; do
2.  printf "%b" "This package's date is $THISPACKAGE\n" >&2
3.  choice "Is that correct? [Y/,<New date>]: "
4.  if [ -z "$CHOICE" ]; then
5.  CHOICE='y'
6.  elif [ "$CHOICE" != "y" ]; then
7.  printf "%b" "Overriding $THISPACKAGEwith ${CHOICE}\n"
8.  THISPACKAGE=$CHOICE
9.  fi
10. done
```

List of options

```
1. # cookbook filename: select_dir
2. directorylist="Finished $(ls /)"
3. PS3='Directory to process? ' # Set a useful select prompt
4. until [ "$directory" == "Finished" ]; do
5. printf "%b" "\a\n\nSelect a directory to process:\n" >&2
6. select directory in $directorylist; do
7. # User types a number which is stored in $REPLY, but select
8. # returns the value of the entry
```

```
9. if [ "$directory" = "Finished" ]; then
10. echo "Finished processing directories."
11. break
12. elif [ -n "$directory" ]; then
13. echo "You chose number $REPLY, processing $directory ..."
14. # Do something here
15. break
16. else
17. echo "Invalid selection!"
18. fi # end of handle user's selection
19. done # end of select a directory
20. done # end of while not finished
```

Prompting for a password

- Example:
  1. read -s -p "password: " PASSWD
  2. printf "%b" "\n"
- Note: the option “-s” : not echo the characters typed