Shell Programming 1

01418235

Chavalit Srisathapornphat

Outline

- Prerequisite
- More on processes
- Simple shell script
- Positional parameters
- Expressions
- if-then-else
- Script termination
- Problems

Prerequisite

- Topics you have to be familiar with
  - Shell startup files
  - . or source – to run a script within the current shell
  - Redirection
  - Pipe
  - Shell variables
    - $PATH and etc.
  - Job controls, foreground/background jobs
  - Separating/Grouping commands
  - Go back and review lecture “Introduction to Shells” and “Filters”

Listing Running Processes

- To find out what commands are running
  - jobs
  - ps –f
    - UID User ID that this process belongs to (the person running it).
    - PID Process ID & PPID Parent process ID
    - C CPU utilization of process.
    - STIME Process start time (when it began).
    - TTY Terminal associated withe the process.
    - TIME Process running time.
    - CMD The command that started this process. CMD with -f is different from C MD without it; itshows any command line options and arguments.
    - Nice value -- used in calculating process priority.

Parent and Child Processes

- PPID
  - Every process has a parent process
  - Each new command is run in a child process
    - Current shell waits for the child process to complete
    - To let the current shell runs a new command
      - Use . or source
  - To let a new command runs in the current process
    - Use exec, the current process will be replaced by the new process (of the new command)
  - To kill a process
    - kill %<job id>
    - kill -<signal_type> process_id

A Simple Shell Script

- What is a shell script?
  - A file that contains commands that the shell can execute
    - Any commands and techniques you've been using at a shell prompt can be used in a script
    - Additional control flow commands can be used
    - Script helps to quickly automating a complex series of tasks or a repetitive procedure
    - (at least) chmod u+x <script> is required to run it

The First Script

```
#!/bin/bash
# My first shell script
date
echo “Users currently logged in”
who
```

- Specifying a shell to run the script
  - The OS invokes the program specified after the first ‘#!’ characters in the script to execute the rest of the script
- Comments
  - Any text after a **#** in any other location until the end of the line
  - The first line starts with a **#!** (shebang) is to tell the shell to invoke the program specified after **#!** to handle the rest of the script

Arguments and Positional Parameters

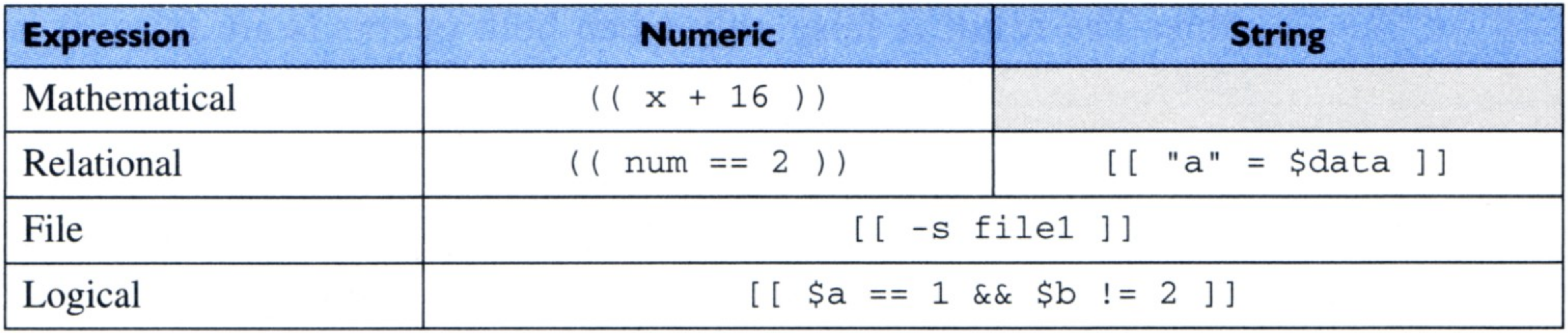
- Positional parameters
  - Command line arguments are put in the positional parameters **$1, $2, …, ${10}, …**

```
#!/bin/bash
# My first shell script
head $1
echo …
tail $1
```

  - **$0** is the command itself
  - **$*** is the list of all arguments

Expressions

- Expressions in shell scripts
  - Mathematical
  - Relational
  - File
  - Logical



Expression	Numeric	String
Mathematical	`(( x + 16 ))`	
Relational	`(( num == 2 ))`	`[[ "a" = $data ]]`
File	`[[ -s file1 ]]`	
Logical	`[[ $a == 1 && $b != 2 ]]`	

Mathematical Expression

- Only integer operations are valid
- Operators
  - **+ - * / % ++ -- ** << >> & | ~**
  - Assignment form is valid e.g. **((var += 2))**
- **let** or double parenthesis

```
#!/bin/bash
# My first shell script
count=5
echo $count
let "count = $count + 5"    # correct way, or
# (( count = $count + 5 ))
echo $count
count=$count+30      # doesn’t work
echo $count
```

Math. Exp. (more examples)

- `c=2; let "c <<= 2"; echo $c`
- `c=64; (( c >>= 2 )); echo $c`
- `c=5; let "c &= 3"; echo $c`
- `c=5; (( c = $c | 3 )); echo $c`
- `c=2; let "c <<= 48"; echo $c`
- `c=1; (( c = ~ $c )); echo $c`

Relational Expressions

- Operators
  - Numeric
    - **`>   >=   <   <=   ==   !=`**
    - Examples

```
a=5
(( a == 5 )); echo $?
(( a >  4 )); echo $?
(( a >= 6 )); echo $?
(( a <  6 )); echo $?
```

  - String
    - **`=`**   **`!=`**   **`-n`** (length is not zero)   **`-z`** (length is zero)
    - Examples

```
s=ab
[[ $s = "ab " ]]; echo $?
[[ $s != "ab" ]]; echo $?
[[ -n $s ]]; echo $?
```

- Note
  - be careful about the space!
  - the result is kept in **`$?`**

File Expressions

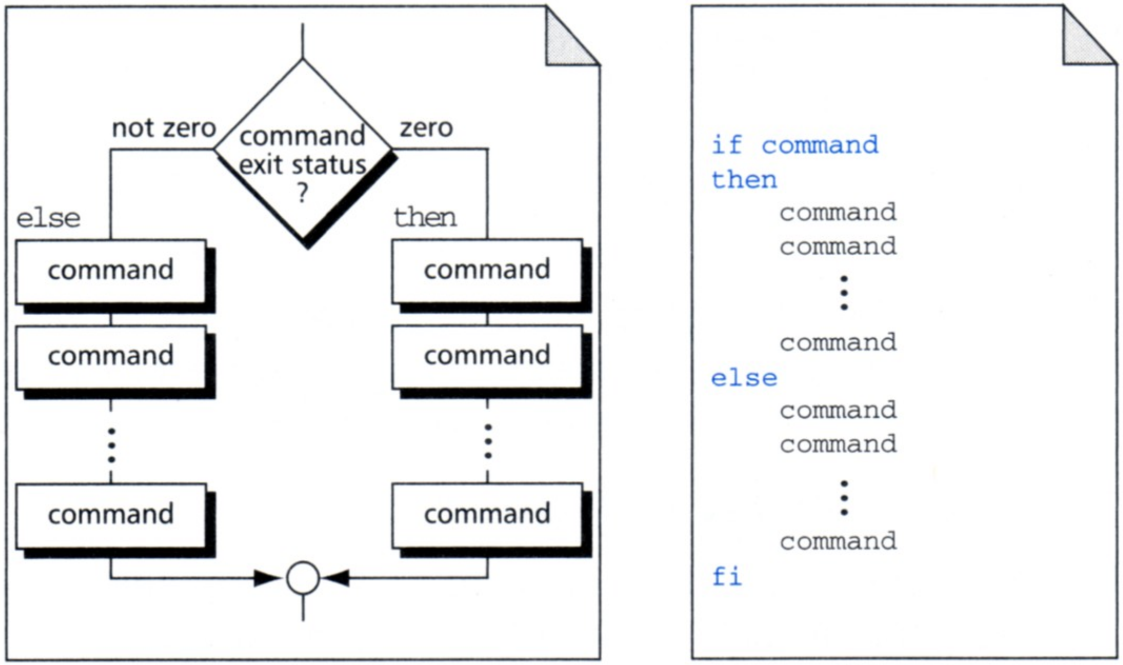
- Operators
  - **-a** – file exists
  - **-r** – file exists and readable
  - **-h** – file exists and is a symbolic links
  - **-w** – file exists and is writable
  - **-x** – file exists and is executable
  - **-f** – file exists and is a regular file
  - **-d** – file exists and is a directory
  - **-s** – file exists and larger than 0 bytes
  - **file1 -nt file2** – file1 is newer than file2
  - **file1 -ot file2** – file1 is older than file2
- Test file command
  - **[[-s file1]]**

Logical Expressions

- Operators
  - **!** (not) **&&** (and) **||** (or)
  - Examples

```
[[ 7 > 5 && 6 > 5 ]]; echo $?
[[ 7 == 5 || 6 < 7 ]]; echo $?
[[ ! "a" = "b" ]]; echo $?
```

Control Flow: if-then-else



(a) Logical Flow

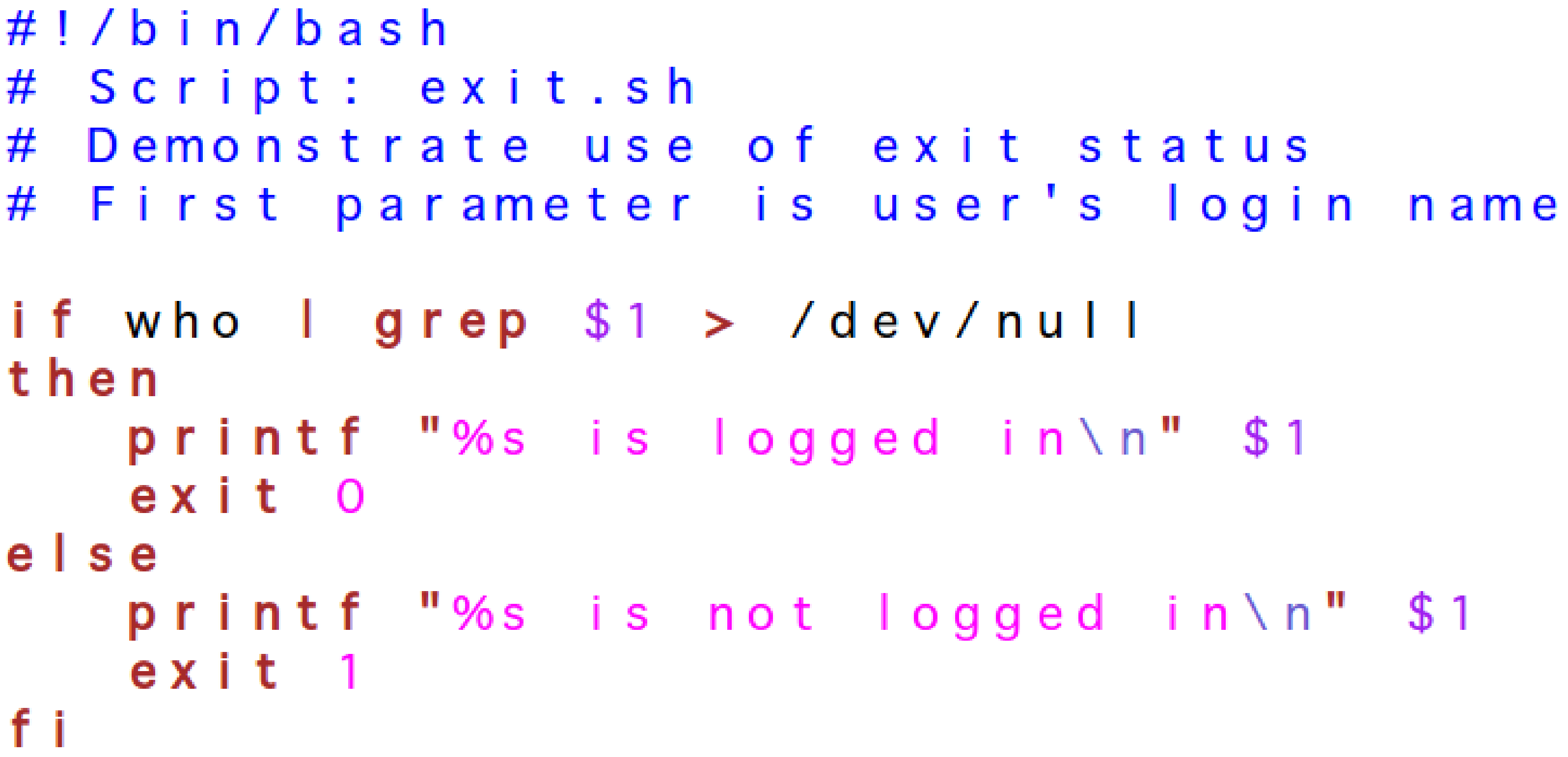
```
if command
then
     command
     command
        :
     command
else
     command
     command
        :
     command
fi
```

(b) Code

- if command evaluates the exit status of the command following if
  - True or 0, commands after then are run
  - False or 1, commands after else are run

Control Flow: if-then-else

- Example

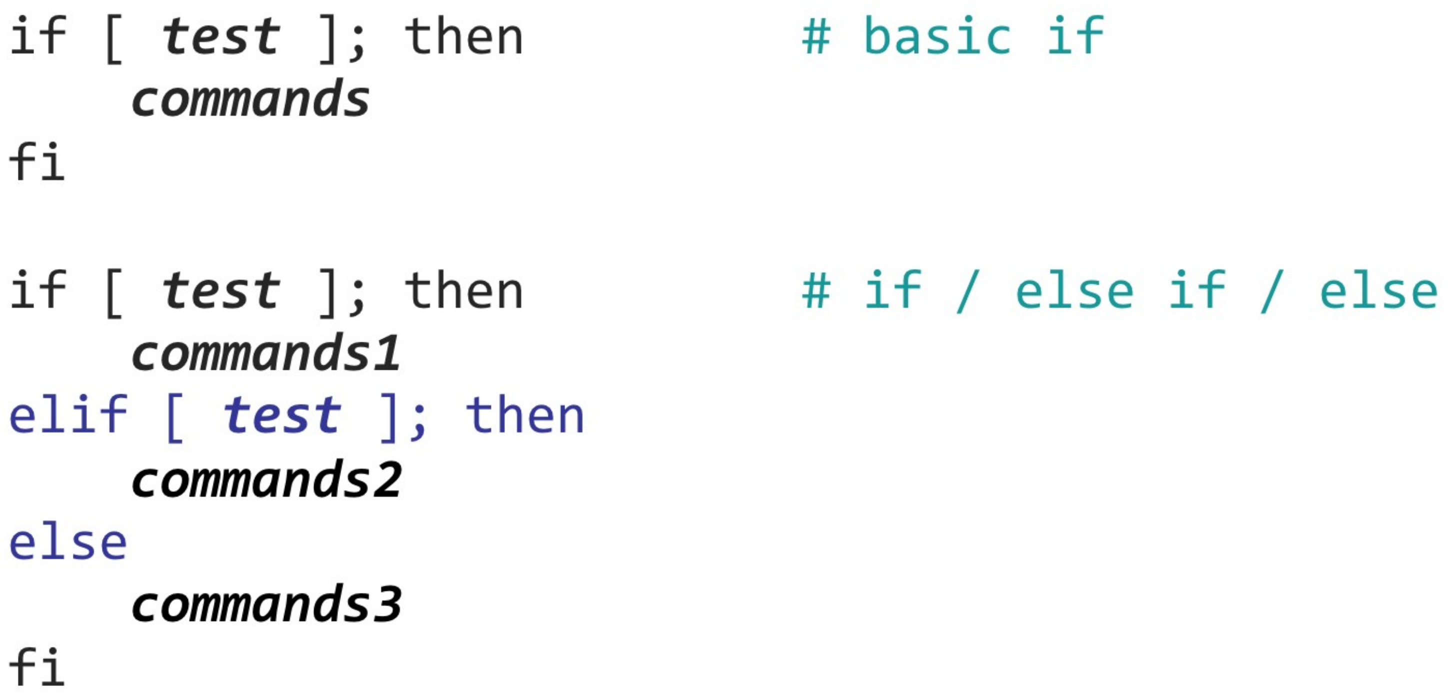


```
#!/bin/bash
# Script: exit.sh
# Demonstrate use of exit status
# First parameter is user's login name

if who | grep $1 > /dev/null
then
    printf "%s is logged in\n" $1
    exit 0
else
    printf "%s is not logged in\n" $1
    exit 1
fi
```

- Check exit status of the command after if

Control Flow: if-then-elif



```
if [ test ]; then              # basic if
     commands
fi

if [ test ]; then              # if / else if / else
     commands1
elif [ test ]; then
     commands2
else
     commands3
fi
```

- [ ] is the 'old' shell test command
- The differences between [ ] and [[ ]]
  - http://mywiki.wooledge.org/BashFAQ/031
- image source:
  - https://slideplayer.com/slide/4951650/

Script Termination

- exit
  - Terminate the script and sets the exit status
  - exit status is kept in system variable $?

Exercise

- Practice set

1. Write a shell script that checks if it is a winter (Nov-Feb), summer (Mar-Jun), or rainy (Jul-Oct) season.
2. Write a script that reads only one argument (as a filename) and tells

   the type of the file (if known),
   "Can't read the file" (if no read permission),
   "Not exist" (if there is no such file) or
   "Unknown" (all other cases).
3. Write a script that reports whether a specific user is running a specific program/command (don't use awk, use ps axo, check out ps's manual)
   Example: **`ps axo user:30,pid,pcpu,pmem,vsz,rss,tty,stat,start,time,comm`**
4. Write a script that reports the total number of times a specific user has logged in to the system (Hint: get the login information from "last -w" command and use awk –f script –v varname=value)