LPIC-1 TRAINING COURSE

Topic 105: Shells and Scripting

Contents

Objectives

- Customize shell environment to meet users's needs
- Modify global and user profiles
- Customize existing scripts or write simple new BASH scripts

1. Customize and use the shell environment

Login vs. Non-login shell

- ***Login shell***: Shell started with **login**, **bash –l** or **su –** command
  - ***Login shell*** reads a series of configuration files as it started
- ***Non-login shell***: Shell started any other way
  - ***Non-login*** shells inherit settings (environment variables) from the parent program with started it
  - Common environment variables: **`PATH, SHELL, PWD, HOME, UID, PS1…`**

bash configuration files

Type of File	Login shell	Non-login shell
Global	**/etc/profile** **/etc/profile.d/***	**/etc/bashrc** or **/etc/bash.bashrc**
User	**~/.bash_profile** or **~/.bash_login** (if **~/.bash_profile** doesn’t exist) or **~/.profile** (if **~/.bash_profile** and **~/.bash_login** doesn’t exist)	**~/.bashrc**
Extra files	**/etc/inputrc, ~/.inputrc, ~/.bash_logout**	

- **/etc/skel** directory holds user files that are copied to individual user’s home directories

Commands for shell variables

- Assign/set a variable: **<u>VARNAME</u>=<u>value</u>**
- Export a variable to be the environment variable:

```
          export VARNAME
    or:   export VARNAME=value
```

- Remove a variable: **unset <u>VARNAME</u>**
- Display all the enviroment variables: **env**

Aliases

- Normally used to create command shortcuts
- Aliases are NOT exportable
- Paramenters added to alias will be added at the end of the real command
- Aliases are often defined in **~/.bashrc** or **~/.bash_profile**
- Alias commands:
  - Displays all the current shell aliases: **`alias`**
  - Sets a new alias: **`alias <u>AliasName</u>="<u>command(s)</u>…"`**
    - Eg: **`alias  dir="ls -al"`**
  - Deletes the alias: **`unalias <u>AliasName</u>`**

Functions

- Functions provide additional capability than alias, including process paramenters
- Functions can be exported
- Variable can be passed-on to functions and will be recognized as **$1**, **$2**, **$3**…

$1-$9	positional parameters
$#	number of positional paramenters
$*	***"$1 $2 $3…"***
$@	***"$1" "$2" "$3" …***

- Functions can return only number

Functions (cont')

- Function declaration:

```
function copyit() {
    echo "Copying" $1 "to" $2
    cp $1 $2
    return $?
}
```

- Use function:

```
copyit /tmp/oldfile /tmp/newfile
```

- Export function: `export –f copyit`
- Delete function: `unset –f copyit`

Command search priority

- **bash** tries to find command in the following sequence:
  1. Aliases
  2. Functions
  3. Builtin commands
  4. Searching the ***PATH***
- To force using a orginial command (command found in the **PATH**), use backslash (\) followed by command
  - Eg: `\ls /tmp`
- To force using a builtin command, use the command ***builtin***
  - Eg: `builtin kill firefox`

Exercise

1. Create an alias for ***wall Hello*** called ***wall***
   - Hint: `alias wall=‘wall Hello’`
2. Try it out with `wall “Good Morning”`
3. Check that `\wall“Good Morning”` revert to it’s normal unaliased behavior
4. Delete the alias with the `unalias` command
   - Hint: `unalias wall`
5. Check that this alias is no longer works
6. Append this alias to your **.bashrc** file
   - Hint: `echo “alias wall=‘wall Hello’ ” >> ~/.bashrc`
7. Recheck your **.bashrc** file for this alias
8. Try it from your current shell (it won’t work yet)
9. Exit current shell and open a new shell and retry it

2. Customize or write simple scripts

What is a shell script

- A text file that tells the shell what to do
- First line contains **#!** and the name of the program that is used as the interpreter
  - Eg:

    ```
    #!/bin/bash
    #!/bin/sh
    #!/usr/bin/perl -w
    ```
- Conditions for running a script:
  - Script file must be runnable by the user running it (**chmod**)
  - The interpreter must be where the script says it is. Default is to call **bash**

Passing parameters to a script

- Scripts can be given up to 9 positional paramenters
  - Up to 99 parameters with **bash**
- Parameters will be identified as **$1** to **$9** or **${10}** to **${99}**

```
scriptname   parm1   parm2   parm3 …   parm66   …
    $0          $1     $2        $3  …    ${66}  …
```

- Parameter **$n** can be modified by set command inside the script

```
    set   value1   value2    value3    …
       $1       $2           $3      …
```

Special Parameters

Parameter	Description
`$n`	Positional paramenter n (max n=9). ***$0*** is the name of shell script
`${nn}`	Positional parameter nn (for nn>9)
`$#`	Number of positional parameters (not including the script)
`$@, $*`	All positional parameters
“#@”	Same as “$1” “$2” … “$n”
“$*”	Same as “$1c$2c…$n” with c is content of $IFS (default is space)
`$?`	Exit status of the last command
`$$`	Process ID of the current shell
`$is`	Name of the current shell
`$!`	Process ID of the last background command

The *shift* command

- **`shift`** moves the assignment of the positional parameters to the left
- Example:

```
$ cat shift_test
#!/bin/bash
echo $1 $2 $3
shift
echo $1 $2 $3
$ ./shift_test aaa bbb ccc ddd
aaa  bbb ccc
bbb ccc  ddd
```

Conditional Expressions

- **test** and **[ ]** command evaluate conditional expression with file attributes, strings and integers
  - Syntax: **`test expression`** or **`[ expression ]`**
- Return status: *zero* (true), *non-zero* (false)
- Example:
  - Test if ***filename*** exist: **`test –a filename`**
  - Test if ***file1*** is newer than ***file2***: **`test file1 –nt file2`**
  - Test if ***string*** is zero: **`test –z string`**
  - Test if ***string1*** is equal to ***string2***:
    `[ string1 == string2 ]`
  - Test if **VAR1** is greater than 4: **`[ $VAR1 –gt 4 ]`**

Conditional Statement: *if*

- **if** allows certain commands to execute only if conditions are met
- Syntax:

```
if <condition_is_true>; then
        commands;
        ….
elseif <condition_is_true>; then
        commands;
        …
else
        commands;
        …
fi
```

- Examples of condition:
  - Test file status: `if test -e /etc/fstab; then`
  - Test command exit code: `if (ps -ef | grep ‘apache’); then`
  - Test contents of a variable: `if $1; then`
  - String testing: `if [ “$mystring” = “hello” ]; then`
  - Integer testing: `if  test “$#” -eq 5; then`

Conditional Statement: *case*

- **case** is normally used for conditionally branching to one of several choices depending on the content of a variable
- Syntax:

```
case <variable> in
<choice1>)
       commands;
;;
<choice2>)
       commands;
;;
 <choice3>)
       commands;
;;
*)
       commands;
;;
esac
```

Looping: *while* loop

- **while** keeps looping and running the commands in its block for as long as its condition(s) is/are met
- Syntax:

```
while <condition_is_true>;
do
    commands;
done
```

Looping: *until* loop

- **until** works exactly the same way as **while** except that the logic is opposite
- Syntax:

```
until <condition_is_true>; do
    commands;
done
```

Looping: *for* loop

- **for** allows a sequence of commands to be execute as many times as there are items in a given list
- Each time the loop runs through, the content of a specific variable becomes value of the current item in the given list
- Syntax:

```
        for variable in list ; do
            commands;
        done
```

- Example:

```
for item in ~/file1 ~/file2 ~/file3 ; do
   echo “-----Content of $item----”
   cat $item
done
```

Shell functions

- Syntax:

```
function FunctionName () {
    commands;
}
```

  or:

```
FunctionName () {
    commands;
}
```

- See *function* in the previous section (*Customize and use the shell environment*) for more detail

Exit codes and the variable *$?*

- All programs, including scripts, return an exit code whe their process ends.
- Exit code can be read via special variable ***$?***
  - Generally exit code of '**0**' means success, other (*1-255*) means some sort of failure
- Exit code normally used to make decision further in the calling script

Localtion and security for *bash* script

- Administration scripts are normally stored in the **PATH** (***/usr/local/bin*** or ***/root/bin***)
- Normal access right are *755 (rwxr-xr-x)* or *700 (rwx------)*
- SUID doesn't have any effect on scripts

Exercise 1

1. Open a terminal from GUI environment
2. Start an editor and tell it to create a file called **testscript** in **/tmp** directory
   - Hint: `vi /tmp/testscript`
3. Type the following lines into the editor:

```
#!/bin/bash
for FILE in `ls *.txt` ; do
     echo -n “Display $FILE ? ”
     read ANSWER
     if [ $ANSWER == ‘y’ ]
     then
             less $FILE
     fi
done
```

4. Be sure you’ve typed every character correctly. One common error is mistyping the back-tick character (`) as single quote character (‘)
5. Save the file and exit the editor
6. Type `chmod a+x /tmp/testscript` to add executable bit to the file’s permissions
7. Type **/tmp/testscript** to run the script. If there is no text (*.txt) files in your current directory, the script displays a `no such file or directory` error message; but if any text files are present, the script gives you the option of viewing each one in turn via `less`.

Exercise 2

1. Create a script to display **`Hello World!`** when run. Test this script.
2. Create a script to display a greeting to the name given as input value. Example: running **`./myscript Long`** will display **`Hello Long`**. Run it.
3. Create a script to ask a user for his name, then display a greeting to the given name (Example: **`Hello Tuan`**). Run it.
4. Create a script that display a list of greetings and ask user to select one. After user selected, clear the terminal and display this greeting only. Run it.
5. Create a script to repeatly display anything input by user. Script will exit when user type **`bye`**. Run it.

Hints to Exercise 2

```
vi myscript1
#!/bin/bash
echo “Hello World!”
chmod a+x myscript1
./myscript1
Hello World!
```

```
vi myscript2
#!/bin/bash
echo “Hello $1”
./myscript2 “Van Anh”
Hello Van Anh
```

```
vi myscript3
#!/bin/bash
echo “What’s your name:”
read NAME
echo “Hello $NAME”
./myscript3
What’s your name:
Thang
Hello Thang
```

```
vi myscript4
#!/bin/bash
echo “1 - Hello”
echo “2 - Good Morning”
echo “3 - Good Night”
echo “What’s your favourite greeting?”
read CHOICE
case $CHOICE in
  1) echo “Hello”;;
  2) echo “Good Morning”;;
  3) echo “Good Night” ;;
esac
./myscript4
```

```
vi myscript5
#!/bin/bash
STRING=nothing
while [ “$STRING” != “bye” ]
do
  echo “Type in your string:”
  read STRING
  echo $STRING
done
./myscript5
```

Thank You !

BACKUP SLIDES