LPIC-1

Exam 102

LINAGORA

Linagora Formation
<formation@linagora.com>

Summary

Summary

Shell

- It is the tool of choice for UNIX.
- There are two main ones:

sh	Bourne shell
csh	C shell
tcsh	TENEX C shell
ksh	Korn shell
bash	Bourne-again shell
zsh	Z shell

- The shell commands by default in Linux test bash (this is the one we will use later).

Shell

http://fr.wikipedia.org/wiki/Shell_Unix
http://explainshell.com/
http://en.wikipedia.org/wiki/Bourne_shell
http://en.wikipedia.org/wiki/C_shell
http://www.tcsh.org/Welcome
http://en.wikipedia.org/wiki/Tcsh http://www.kornshell.com/
http://en.wikipedia.org/wiki/Korn_shell
http://www.gnu.org/software/bash/ http://fr.wi
kipedia.org/wiki/Bourne-Again_shell
http://zsh.sourceforge.net/ http://fr.wikipedia.org/wiki/Zsh
http://fr.wikipedia.org/wiki/Shellshock_(faille_informatique)

The variables

Principle

- A variable can store a value to future use.
- The shell has only one variable, which stores strings (where some programming languages distinguish several types of variables based on the values they can store).
- There are three categories of variables:
- Special variables, whose name consists of a symbol or a figure whose value is set by the shell;
    - Environment variables, whose name is traditionally made up of capitals and whose value is fixed in the configuration of the command is inherited from his father;
    - Variables defined by the user, whose name is traditionally made up of letters in lower case.

The variables

Creating and initializing a variable defined by the user

- The name of a variable defined by the user can contain letters (it makes the difference between capitals and lower case), numbers or underscores. It can not start with a number.
- A variable is created by assigning a value using the following syntax:

```
$ name=value
```

We must put absolutely no space around the equal sign.

- If the value contains spaces, each of which must be preceded by a backslash or the value must be surrounded by double quotation marks or apostrophes:

```
$ name=value\ with\ spaces
$ name="value with spaces"
$ name='value with spaces'
```

The variables

Value of a variable

- The shell replaces any dollar sign $ followed immediately by a variable name by the value of this variable.
- We can for example display the value of a variable using the echo command with argument in a symbol immediately followed dollar variable name:

```
$ echo $name
value
```

The variables

The command `set`

- The command `set` displays:
- the environment variables;
- variables defined by the user;
- the function definitions.

Each line of the display indicates the name of a variable followed by an equal sign followed by the value of the variable:

```
$ set

[...]
name=value
[...]
```

http://www.gnu.org/software/bash/manual/html_node/The-Set-Builtin.html

The variables

The command env

- The command env displays the environnement variables .
  Each line of the display indicates the name of a variable followed by an equal sign followed by the value of the variable:

```
$ env

[...]
HOME=/home/formation
LOGNAME=formation
PATH=/usr/local/bin:/usr/bin:/bin
SHELL=/bin/bash
TERM=xterm
USER=formation
 [...]
```

The variables

Some classic environment variables

DISPLAY	access path through the X server : $ echo $DISPLAY :0.0
HOME	absolute access path to the user's home directory: $ echo $HOME /home/formation
LANG	language to use for display $ echo $LANG fr_FR.UTF-8

http://fr.wikipedia.org/wiki/Variable_d'environnement

The variables

Some classic environment variables

PATH — list of search paths for commands:

```
$ echo $PATH
PATH=/usr/local/bin:/usr/bin:/bin
```

HOME — absolute access path of the current directory:

```
$ echo $PWD
/home/formation
```

LANG — absolute access path of the shell:

```
$ echo $SHELL
/bin/bash
```

USER — user login:

```
$ echo $USER  formation
```

The variables

The command export

- Transform any variable environment variable:

```
$ NAME=value
$ export NAME
```

- Both operations can be performed at once:

```
$ export NAME=value
```

The variables

Temporary environment variables

- It is possible to create environment variables that exist only during the execution of an order.
- For this, we can use two syntaxes:
  - by preceding the control (with its options and possible arguments) to initialize environment variables:

    ```
    $ NAME1=value1 NAME2=value2 command
    ```

  - using the env command and an identical formulation:

    ```
    $ env NAME1=value1 NAME2=value2 command
    ```

The variables

The command unset

- Destroy a variable:

```
$ echo $name
valeur
$ unset nom
$ echo $nom  [ligne
blanche]
```

http://en.wikipedia.org/wiki/Unset

Search orders

- When an order is entered simply by name, the corresponding executable is searched for in the directories specified by the PATH environment variable.
- We can also run a command with its path (absolute or relative):

```
$ /bin/pwd
/home/formation
```

http://en.wikipedia.org/wiki/PATH_(variable)

The aliases

- An alias is a new order built from an existing command, often used with specific options.
- Without argument, the alias command displays the list of existing aliases:

```
$ alias  [...]
```

- With an argument of the form name=definition, the alias command defines an alias called name:

```
$ alias ll='ls -l'
```

http://fr.wikipedia.org/wiki/Alias_(informatique)
http://www.gnu.org/software/bash/manual/html_node/Aliases.html

The aliases

- If an alias has the same name as an existing command (which is possible), precede the name with a backslash (\) is used to execute the original command:

  ```
  $ \rm toto
  ```

- Finally, unalias to remove an alias:

  ```
  $ unalias ll
  ```

The functions

- A function is a new command to group a set of commands.
- A function is defined and used like this:

```
$ function ()
> {
> echo this is
> echo a function
> }
$ function
this is
a function
```

http://www.gnu.org/software/bash/manual/html_node/Shell-Functions.html

The exit status of a command

Principle

- Just before finishing its execution, any command returns to the command a number indicating whether it was successful or not. This number is the command exit status.
- The exit status of the command that has been executed is stored in the special variable?
- Generally, the exit status is:

  0 if the command was successful

  1 on the other hand

  Some commands may return other values greater than 1 to categorize errors more precisely.

http://en.wikipedia.org/wiki/Exit_status#Bash_shell_and_script

The exit status of a command

Example

```
$ which ls
/bin/ls
$ echo $?
0
$ which glub
$ echo $?
1
```

Succession of commands

Principle

- It is possible to use a command exit status to run another command conditionally.

Succession of commands

Operator &&

- Example :

```
$ command1 && command2
```

- The command command2 will be executed only if the exit status of the command1 command is zero, that is to say, if it is successful. So:
  - if command1 is completed successfully, command2 will be executed;
  - if command1 does not complete successfully, command2 will not be executed.
- For this reason, the operator && is called operator "and".

The spaces around the operator && is not essential.

Succession of commands

Operator ||

- Example :

  ```
  $ command1 || command2
  ```

- La commande commande2 ne sera exécutée que si le statut de sortie de la commande commande1 est non nul, c'est-à-dire si elle ne s'est pas correctement terminée. Donc :
  - si commande1 se termine correctement, commande2 ne sera pas exécutée ;
  - si commande1 ne se termine pas correctement, commande2 sera exécutée.
- Pour cette raison, l'opérateur || est appelé opérateur « ou ».
- Les espaces autour de l'opérateur || ne sont pas indispensables.

Succession of commands

Operator ;

- Example :

  ```
  $ command1 ; command2
  ```

- The separator ; executes a sequence of commands without condition. Here command1 will be executed, followed by command2.
- The spaces around the separator ; are not essential.

Configuration files bash

Principle

bash uses many configuration files, subject to their existence.

http://www.gnu.org/software/bash/manual/html_node/Bash-Startup-Files.html

Configuration files bash

Description

- Invoked as interpreter interactive login shell, bash executes the file:
  - `/etc/profile` (also used by the Bourne shell `sh`)

  then the first of the following files exist:
  - `~/.bash_profile`
  - `~/.bash_login`
  - `~/.profile` (also used by the Bourne shell `sh`)

  When it ends, bash execute the file:
  - `~/.bash_logout`
- Invoked as interpreter of interactive non-login shell, bash executes the file:
  - `/etc/bash.bashrc`
  - `~/.bashrc`

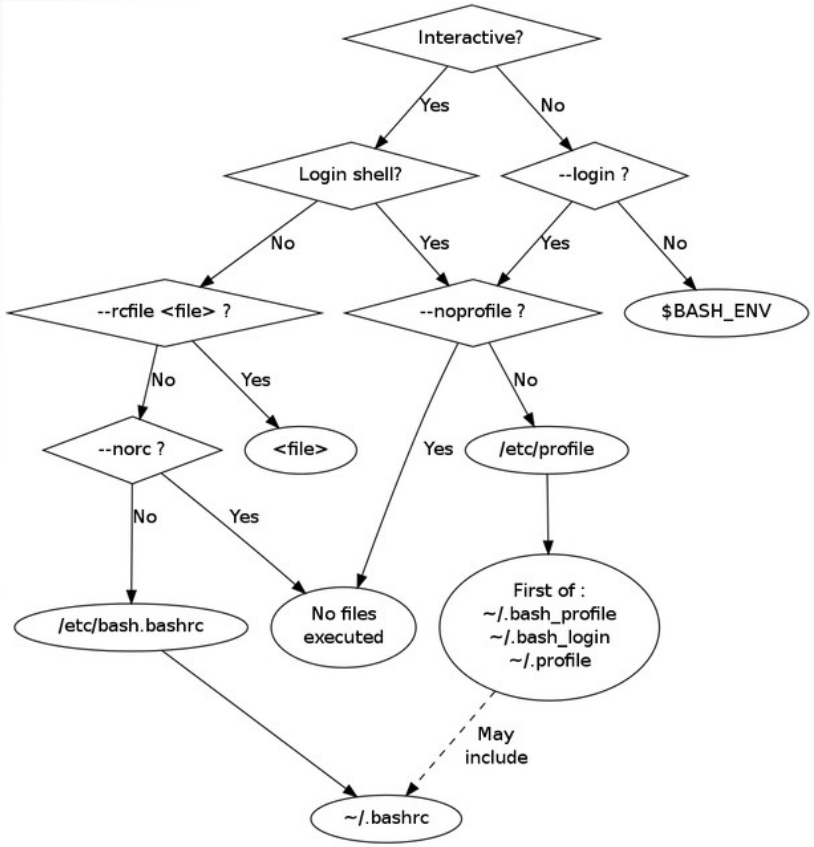
Configuration files bash

Summary

file	*interactive login shell*	*interactive non-login shell*
/etc/profile	✦	
~/.bash_profile	①	
~/.bash_login	②	
~/.profile	③	
/etc/bash.bashrc		✦
~/.bashrc		✦
~/.bash_logout	✦	

Configuration files bash

Chart



http://www.solipsys.co.uk/new/BashInitialisationFiles.html

Configuration files bash

In practice

- The configurations of interactive login shell or mere interpreters are in practice are usually little different. Consequently :
  - at level system, the /etc/bash.bashrc file is often run from the /etc/profile (or the reverse);
  - in the configuration of users, the ~/.bashrc is often executed from the file ~/.profile (or the reverse).

Configuration files bash

Examples of use

- Enable or disable an option bash:

```
$ set -o noclobber
```

- Edit an environment variable:

```
$ export PATH=$PATH:$HOME/bin
```

- Define an alias:

```
$ alias ll='ls -l'
```

- Edit the file creation mask:

```
$ umask 077
```

- Call the command fortune (from one of the files read by an interpreter of interactive login shell) to display a random message early in the session.

http://fr.wikipedia.org/wiki/Fortune_(programme)

The commands . (point) and source

- The commands . (point) and source allow execute the contents of a file by the shell.
- They are therefore well suited to proofreading a configuration file (rather than having to quit and restart in the shell).
- Examples :

```
$ . ~/.bashrc
```

or:

```
$ source ~/.bashrc
```

http://www.gnu.org/software/bash/manual/html_node/Bourne-Shell-Builtins.html#index-_002e
http://www.gnu.org/software/bash/manual/html_node/Bash-Builtins.html#index-source

Summary

Introduction

- The shell doesn't only allow the capture and execution of independent controls, it contains a real programming language.
- Due to their size larger than that of a simple command, and to allow reusability, programs are often seized in files.
- These files are called shell scripts or simply scripts.
- They have a .sh extension when they are to be executed by the Bourne shell sh (the general case), a .bash extension when using specific features of the Bourne-again shell bash or a .csh extension when they use the programming language C shell (which is totally different).

First script

- It can be used in a script all the commands and syntax studied until now.
- A simple command is already a script. Consider the file `premier.sh` suivant :

`file first.sh`

```
echo 'this is my first script'
```

You can run it by providing its path argument of the Bourne shell sh:

```
$ sh first.sh
this is my first script
```

Shebang

Principle

- It would be useful to run a script directly without having to specify which shell to use:

```
$ ./first.sh
```

- To this must be added the first line of the script:

```
#! /bin/sh
```

Resulting in total:

```
#! /bin/sh

echo 'this is my first script'
```

and make the script executable:

```
$ chmod 755 first.sh
```

Shebang

Running

- Les deux caractères `#!` constituent ce qu'on appelle en anglais *hash-bang* ou *sharp-bang* et qui donne par contraction : shebang.
- The two characters `#!` constitute what is called in English *hash-bang* or *sharp-bang* and gives "shebang".
- Lorsqu'un fichier exécutable commence par ces deux caractères, c'est en réalité le chemin d'accès absolu qui suit qui est exécuté, auquel est passé en argument le chemin d'accès sous lequel le fichier exécutable a été appelé.
- Dans notre exemple, la commande :

```
$ ./premier.sh
```

est donc traduite par :

```
$ /bin/sh ./premier.sh
```

http://fr.wikipedia.org/wiki/Shebang

Continuation de ligne

fichier continuation.sh

```
#! /bin/sh

echo 'une barre oblique inverse en fin de ligne'          \
        'permet de\npoursuivre une longue commande'       \
        'sur la ligne suivante'
```

affichage

```
une barre oblique inverse en fin de ligne permet de
poursuivre une longue commande sur la ligne suivante
```

Commentaires

fichier commentaires.sh

```
#! /bin/sh

# les commentaires commencent par un #
# et s'étendent jusqu'à la fin de la ligne

echo 'ce qui suit est un commentaire' # non interprété
```

affichage

```
ce qui suit est un commentaire
```

Merci pour votre attention !