Dernière mise-à-jour : 2020/01/30 03:28

Command Line Interface

The Shell

A shell is a **Command Line Interpreter** (C.L.I). It is used to give instructions or **commands** to the operating system (OS).

The word shell is generic. There are many shells under Unix and Linux such as:

Shell	Name	Release Date	Inventer	Command	Comments
tsh	Thompson Shell	1971	Ken Thompson	sh	The first shell
sh	Bourne Shell	1977	Stephen Bourne	sh	The shell common to all Unix and Linux OSs: /bin/sh
csh	C-Shell	1978	Bill Joy	csh	The BSD shell: /bin/csh
tcsh	Tenex C-Shell	1979	Ken Greer	tcsh	A fork of the csh shell: /bin/tcsh
ksh	Korn Shell	1980	David Korn	ksh	Open Source since 2005: /bin/ksh
bash	Bourne Again Shell	1987	Brian Fox	bash	The default shell for Linux, MacOS X, Solaris 11: /bin/bash
zsh	Z Shell	1990	Paul Falstad	zsh	Zsh is an extended Bourne shell with a large number of improvements, including some features of bash, ksh, and tcsh: /usr/bin/zsh

When using Ubuntu 16.04 **/bin/sh** is a soft link to **/bin/dash** :

```
trainee@ubuntu1604:~$ ls -l /bin/sh
lrwxrwxrwx 1 root root 4 mai    3  2016 /bin/sh -> dash
```

/bin/bash

This unit covers the /bin/bash shell. The **/bin/bash** shell allows you to:

- Recall previously typed commands
- Auto-generate the end of a file name
- Use Aliases
- Use tables
- Use C language numerical and math variables
- Manage strings
- Use Functions

A command always starts with a keyword. This keyword is interpreted by the shell, in the order shown, as one of the following:

- An Alias,
- A Function,
- A Built-in Command,
- An External Command.

Internal And External Commands

The /bin/bash shell comes with a set of built-in or *internal* commands. External commands are executable binaries or scripts generally found in one of the following directories:

- /bin,
- /sbin,
- /usr/bin,
- /usr/sbin.

To check if a command is internal to the shell or external, use the **type** command:

```
trainee@ubuntu1604:~$ type cd
cd is a shell builtin
```

External commands are either binaries or scripts that can be found in /bin, /sbin, /usr/bin or /usr/sbin :

```
trainee@ubuntu1604:~$ type passwd
```

```
passwd is /usr/bin/passwd
```

Aliases

Aliases are strings that are aliased to a command, a command and some options or even several commands. Aliases are specific to the shell in which they are created and unless specified in one of the start-up files, they disappear when the shell is closed:

```
trainee@ubuntu1604:~$ type ls
ls is aliased to `ls --color=auto'
```

[stextbox id='black' image='null'] **Important**: Note that the **ls** alias is an alias to the **ls** command itself. [/stextbox]

An alias is defined using the **alias** command:

```
trainee@ubuntu1604:~$ alias dir='ls -l'
trainee@ubuntu1604:~$ dir
total 48
-rw-rw-r-- 1 trainee trainee    0 oct.   4 14:24 aac
-rw-rw-r-- 1 trainee trainee    0 oct.   4 14:24 abc
-rw-rw-r-- 1 trainee trainee    0 oct.   4 14:24 bca
drwxr-xr-x 2 trainee trainee 4096 mai    3  2016 Desktop
drwxr-xr-x 2 trainee trainee 4096 mai    3  2016 Documents
drwxr-xr-x 2 trainee trainee 4096 mai    3  2016 Downloads
-rw-r--r-- 1 trainee trainee 8980 mai    3  2016 examples.desktop
drwxr-xr-x 2 trainee trainee 4096 mai    3  2016 Music
drwxr-xr-x 2 trainee trainee 4096 mai    3  2016 Pictures
drwxr-xr-x 2 trainee trainee 4096 mai    3  2016 Public
drwxr-xr-x 2 trainee trainee 4096 mai    3  2016 Templates
drwxr-xr-x 2 trainee trainee 4096 mai    3  2016 Videos
-rw-rw-r-- 1 trainee trainee  442 sept. 30 11:35 vitext
-rw-rw-r-- 1 trainee trainee    0 oct.   4 14:24 xyz
```

[stextbox id='black' image='null'] **Important**: Note that **dir** exists as a command. By creating an alias of the same name, the alias will be executed in

place of the command. [/stextbox]

The list of currently defined aliases is obtained by using the **alias** command with no options:

```
trainee@ubuntu1604:~$ alias
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed
-e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')"'
alias dir='ls -l'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

[stextbox id='black' image='null'] **Important**: In the above list you can see, without distinction, the system wide aliases created by system start up scripts and the user created alias **dir**. The latter is only available for trainee and will disappear when the current session is terminated. [/stextbox]

To force the shell to use the command and not the alias, you can precede the command with the **** character:

```
trainee@ubuntu1604:~$ \dir
aac  bca      Documents  examples.desktop  Pictures  Templates  vitext
abc  Desktop  Downloads  Music             Public    Videos     xyz
```

To delete an alias, simply use the **unalias** command:

```
trainee@ubuntu1604:~$ unalias dir
trainee@ubuntu1604:~$ dir
aac  bca      Documents  examples.desktop  Pictures  Templates  vitext
abc  Desktop  Downloads  Music             Public    Videos     xyz
```

Each user's shell is defined by root in the **/etc/passwd** file:

```
trainee@ubuntu1604:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:103:systemd Network Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false
syslog:x:104:108::/home/syslog:/bin/false
_apt:x:105:65534::/nonexistent:/bin/false
messagebus:x:106:110::/var/run/dbus:/bin/false
uuidd:x:107:111::/run/uuidd:/bin/false
lightdm:x:108:114:Light Display Manager:/var/lib/lightdm:/bin/false
whoopsie:x:109:116::/nonexistent:/bin/false
avahi-autoipd:x:110:119:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false
avahi:x:111:120:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/bin/false
colord:x:113:123:colord colour management daemon,,,:/var/lib/colord:/bin/false
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/false
```

```
hplip:x:115:7:HPLIP system user,,,:/var/run/hplip:/bin/false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/bin/false
pulse:x:117:124:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
trainee:x:1000:1000:trainee,,,:/home/trainee:/bin/bash
sshd:x:121:65534::/var/run/sshd:/usr/sbin/nologin
```

However, each user can change his shell using the **chsh** command. The shells available to users are listed in the **/etc/shells** file:

```
trainee@ubuntu1604:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/dash
/bin/bash
/bin/rbash
/usr/bin/screen
```

Now use the **echo** command to view the contents of the system variable SHELL for your current session:

```
trainee@ubuntu1604:~$ echo $SHELL
/bin/bash
```

Now change your shell to **/bin/sh** using the **chsh** command:

```
trainee@ubuntu1604:~$ chsh
Password: trainee
Changing the login shell for trainee
Enter the new value, or press ENTER for the default
    Login Shell [/bin/bash]: /bin/sh
```

[stextbox id='black' image='null'] **Important**: Note that the password will not be printed to standard output. [/stextbox]

Now check your current shell:

```
trainee@ubuntu1604:~$ echo $SHELL
/bin/bash
```

At first glance nothing has happened. However if you view your entry in the **/etc/passwd** file you will notice that your login shell has changed:

```
trainee@ubuntu1604:~$ cat /etc/passwd | grep trainee
trainee:x:1000:1000:trainee,,,:/home/trainee:/bin/sh
```

[stextbox id='black' image='null'] **Important** : The /bin/sh shell will be your active shell the next time you login. [/stextbox]

Now change your shell back to **/bin/bash** using the **chsh** command:

```
trainee@ubuntu1604:~$ chsh
Password: trainee
Changing the login shell for trainee
Enter the new value, or press ENTER for the default
    Login Shell [/bin/sh]: /bin/bash
```

[stextbox id='black' image='null'] **Important**: Note that the password will not be printed to standard output. [/stextbox]

The Prompt

As you have already noticed, the **prompt** under Linux is different for a normal user and root:

- **$** for a user,
- **#** for root.

The history Command

/bin/bash keeps track of commands that have been previously executed. To access the *command history*, use the following command:

```
trainee@ubuntu1604:~$ history | more
    1  sudo su -
    2  vi vitext
    3  view vitext
    4  vi vitext
    5  vi .exrc
    6  vi vitext
    7  clear
    8  stty -a
    9  date
   10  locale
   11  LANG=en_GB.UTF-8
   12  export LANG
   13  locale
   14  date
   15  LC_ALL=en_GB.UTF-8
   16  export LC_ALL
   17  locale
   18  date
   19  who
   20  df
   21  df -h
   22  free
   23  free -h
--More--
```

[stextbox id='black' image='null'] **Important**: The history is specific to each user. [/stextbox]

The history command uses **emacs** style control characters. As a result you can navigate through the list as follows:

Control Character	**Action**
[CTRL]-[P] (= Up Arrow)	Navigates backwards through the list
[CTRL]-[N] (= Down Arrow)	Navigates forwards through the list

To move around in the history:

Control Character	**Action**
[CTRL]-[A]	Move to the beginning of the line
[CTRL]-[E]	Move to the end of the line
[CTRL]-[B]	Move one character to the left
[CTRL]-[F]	Move one character to the right
[CTRL]-[D]	Delete the character under the cursor

Pour rechercher dans l'historique il convient d'utiliser les touches :

Control Character	**Action**
[CTRL]-[R] *string*	Search backwards for *string* in the history. Using [CTRL]-[R] again will search for the previous occurence of *string*
[CTRL]-[S] *string*	Search forwards for *string* in the history. Using [CTRL]-[S] again will search for the next occurence of *string*
[CTRL]-[G]	Quit the search mode

It is also possible to recall the last command executed by using the **!!** characters:

```
trainee@ubuntu1604:~$ ls
aac  bca      Documents  examples.desktop  Pictures  Templates  vitext
abc  Desktop  Downloads  Music             Public    Videos     xyz
trainee@ubuntu1604:~$ !!
ls
aac  bca      Documents  examples.desktop  Pictures  Templates  vitext
abc  Desktop  Downloads  Music             Public    Videos     xyz
```

Alternatively, to execute a command in the list, you can use the list number preceded by the **!** character:

```
trainee@ubuntu1604:~$ !107
ls
aac  bca      Documents  examples.desktop  Pictures  Templates  vitext
abc  Desktop  Downloads  Music             Public    Videos     xyz
```

The environmental variables associated with the history are set in the **~/.bashrc** file where **~/** indicates the home directory of the user concerned:

```
trainee@ubuntu1604:~$ cat .bashrc | grep HISTSIZE
# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
```

As you can see, in the previous case the **HISTSIZE** value is set to **1000**. This means that the last 1,000 commands are held in the history.

The history command stores data in the **~/.bash_history** file for each user. The commands for the current bash session are stored in the file when the session is closed:

```
trainee@ubuntu1604:~$ nl .bash_history | more
     1  sudo su -
     2  vi vitext
     3  view vitext
     4  vi vitext
     5  vi .exrc
     6  vi vitext
     7  clear
     8  stty -a
     9  date
    10  locale
    11  LANG=en_GB.UTF-8
    12  export LANG
    13  locale
    14  date
    15  LC_ALL=en_GB.UTF-8
    16  export LC_ALL
    17  locale
    18  date
    19  who
    20  df
    21  df -h
    22  free
```

```
    23  free -h
--More--
```

[stextbox id='black' image='null'] **Important** : Note the use of the **nl** command to number the lines in the output of the contents of **.bash_history** file. [/stextbox]

The TAB key

/bin/bash can auto-generate the end of a file name. Consider the following example:

```
$ ls .b [Tab][Tab][Tab]
```

By hitting the Tab key three times, the system shows you the files that match:

```
trainee@ubuntu1604:~$ ls .bash
.bash_history  .bash_logout   .bashrc
```

This same technique can also be used to auto-generate command names. Consider the following example:

```
$ mo [Tab][Tab]
```

By hitting the Tab twice the system lists all known commands available to the user and starting with **mo**:

```
trainee@ubuntu1604:~$ mo
moc               mogrify-im6       mount             mount.ntfs
modinfo           montage           mountall          mount.ntfs-3g
modprobe          montage-im6       mount.fuse        mountpoint
mogrify           more              mount.lowntfs-3g  mousetweaks
```

Metacharacters

It is often necessary and desirable to be able to work with several files at one time as opposed to repeating the operation on each file individually. For this reason, bash accepts the use of Metacharacters:

Metacharacter	**Description**
*	Matches one or more characters
?	Matches a single character
[abc]	Matches any one of the characters between square brackets
[!abc]	Matches any character except those between square brackets
[m-t]	Matches any character from m through to t
[!m-t]	Matches any character other than m through to t
?(expression1|expression2| ...)	Matches 0 or 1 occurence of expression1 OR 0 or 1 occurence of expression2 OR ...
***(expression1|expression2| ...)**	Matches 0 to x occurences of expression1 OR 0 to x occurences of expression2 OR ...
+(expression1|expression2| ...)	Matches 1 to x occurences of expression1 OR 1 to x occurences of expression2 OR ...
@(expression1|expression2| ...)	Matches 1 occurrence of expression1 OR 1 occurence of expression2 OR ...
!(expression1|expression2| ...)	Matches 0 occurrences of expression1 OR 0 occurrences of expression2 OR ...

To illustrate the use of Metacharacters, you need to create a directory in your home directory and the create some files within it:

```
trainee@ubuntu1604:~$ mkdir training
trainee@ubuntu1604:~$ cd training
trainee@ubuntu1604:~/training$ touch f1 f2 f3 f4 f5
```

The * Metacharacter

Now use the Metacharacter *:

```
trainee@ubuntu1604:~/training$ echo f*
f1 f2 f3 f4 f5
```

[stextbox id='black' image='null'] **Important**: Note that the * is used as a wild card which replaces 0 or more characters. [/stextbox]

The ? Metacharacter

Create two more files:

```
trainee@ubuntu1604:~/training$ touch f52 f62
```

Now use the Metacharacter **?**:

```
trainee@ubuntu1604:~/training$ echo f?2
f52 f62
```

[stextbox id='black' image='null'] **Important**: Note that the **?** is used as a wild card which replaces a single character. [/stextbox]

The [] Metacharacter

The [] Metacharacter can take several forms:

Metacharacter	Description
[xyz]	Represents either x or y or z
[m-t]	
[!xyz]	Represents any character other than x or y or z
[!m-t]	Represents any character outside of the range m to t

To demonstrate the use of the metacharacter [], create a file called **a100**:

```
trainee@ubuntu1604:~/training$ touch a100
```

The use of this Metacharacter can be demonstrated with the following examples:

```
trainee@ubuntu1604:~/training$ echo [a-f]*
a100 f1 f2 f3 f4 f5 f52 f62
```

```
trainee@ubuntu1604:~/training$ echo [af]*
a100 f1 f2 f3 f4 f5 f52 f62
```

[stextbox id='black' image='null'] **Important**: Note that all the files starting with either **a**, **b**, **c**, **d**, **e** or **f** are displayed. [/stextbox]

```
trainee@ubuntu1604:~/training$ echo [!a]*
f1 f2 f3 f4 f5 f52 f62
```

[stextbox id='black' image='null'] **Important**: Note that all the files in the directory are displayed except the file starting with **a** . [/stextbox]

```
trainee@ubuntu1604:~/training$ echo [a-b]*
a100
```

[stextbox id='black' image='null'] **Important**: Note that only the file starting with **a** is displayed since no file starting with **b** is present. [/stextbox]

```
trainee@ubuntu1604:~/training$ echo [a-f]
[a-f]
```

[stextbox id='black' image='null'] **Important**: Note that in the above example, since no file called **a**, **b**, **c**, **d**, **e** or **f** exists in the directory, the **echo** command simply returns the filter used. [/stextbox]

The extglob Option

In order to use **?(expression), *(expression), +(expression), @(expression) and !(expression)**, you need to activate the **extglob** option:

```
trainee@ubuntu1604:~/training$ shopt -s extglob
```

The **shopt** command is used to activate and deactivate the shopt option of the shell.

The list of all the options can be displayed by simply using the **shopt** command:

```
trainee@ubuntu1604:~/training$ shopt
```

```
autocd          off
cdable_vars     off
cdspell         off
checkhash       off
checkjobs       off
checkwinsize    on
cmdhist         on
compat31        off
compat32        off
compat40        off
compat41        off
compat42        off
complete_fullquote  on
direxpand       off
dirspell        off
dotglob         off
execfail        off
expand_aliases  on
extdebug        off
extglob         on
extquote        on
failglob        off
force_fignore   on
globstar        off
globasciiranges off
gnu_errfmt      off
histappend      on
histreedit      off
histverify      off
hostcomplete    off
huponexit       off
interactive_comments    on
lastpipe        off
lithist         off
```

```
login_shell     on
mailwarn        off
no_empty_cmd_completion off
nocaseglob      off
nocasematch     off
nullglob        off
progcomp        on
promptvars      on
restricted_shell    off
shift_verbose   off
sourcepath      on
xpg_echo        off
```

?(expression)

Create the following files:

```
trainee@ubuntu1604:~/training$ touch f f.txt f123.txt f123123.txt f123123123.txt
```

Execute the following command:

```
trainee@ubuntu1604:~/training$ ls f?(123).txt
f123.txt  f.txt
```

[stextbox id='black' image='null'] **Important**: Note that the command displays file names that match 0 or 1 occurrences of the string **123**. [/stextbox]

***(expression)**

Execute the following command:

```
trainee@ubuntu1604:~/training$ ls f*(123).txt
```

```
f123123123.txt  f123123.txt  f123.txt  f.txt
```

[stextbox id='black' image='null'] **Important**: Note that the command displays file names that match 0 to x occurrences of the string **123**. [/stextbox]

+(expression)

Execute the following command:

```
trainee@ubuntu1604:~/training$ ls f+(123).txt
f123123123.txt  f123123.txt  f123.txt
```

[stextbox id='black' image='null'] **Important**: Note that the command displays file names that match 1 to x occurrences of the string **123**.. [/stextbox]

@(expression)

Execute the following command:

```
trainee@ubuntu1604:~/training$ ls f@(123).txt
f123.txt
```

[stextbox id='black' image='null'] **Important**: Note that the command displays file names that match 1 occurrence of the string **123**. [/stextbox]

!(expression)

Execute the following command:

```
trainee@ubuntu1604:~/training$ ls f!(123).txt
f123123123.txt  f123123.txt  f.txt
```

[stextbox id='black' image='null'] **Important**: Note that the command displays file names that match 0 or x occurrences of the string **123**, where x>1. [/stextbox]

Protecting Metacharacters

To cancel the wild card effect of a special character, the character needs to be escaped or “protected”:

Character	Description
\	Escapes the character which immediately follows
' '	Protects any character between the two '
“ ”	Protects any character between the two **“** except the following: **$**, **** and **'**

For example:

```
trainee@ubuntu1604:~/training$ echo * est un caractère spécial
a100 f f1 f123123123.txt f123123.txt f123.txt f2 f3 f4 f5 f52 f62 f.txt est un caractère spécial

trainee@ubuntu1604:~/training$ echo \* est un caractère spécial
* est un caractère spécial

trainee@ubuntu1604:~/training$ echo "* est un caractère spécial"
* est un caractère spécial

trainee@ubuntu1604:~/training$ echo '* est un caractère spécial'
* est un caractère spécial
```

Exit Status

Each command returns an **exit status** when it is executed. This exit status is stored in a special variable: **$?**.

For example:

```
trainee@ubuntu1604:~/training$ cd ..
trainee@ubuntu1604:~$ mkdir codes
trainee@ubuntu1604:~$ echo $?
0
trainee@ubuntu1604:~$ touch codes/exit.txt
trainee@ubuntu1604:~$ rmdir codes
rmdir: failed to remove 'codes': Directory not empty
trainee@ubuntu1604:~$ echo $?
1
```

As you can see when the exit status is 0, the command has executed correctly. If the exit status is anything else, the command has executed with errors.

Redirections

Your dialogue with the system uses three **file descriptors**:

- Standard Input - the keyboard,
- Standard output - the screen,
- Standard error - contains any eventual errors.

The standard output can be redirected using the **>** character:

```
trainee@ubuntu1604:~$ pwd
/home/trainee
trainee@ubuntu1604:~$ cd training
trainee@ubuntu1604:~/training$ free > file
trainee@ubuntu1604:~/training$ cat file
              total        used        free      shared  buff/cache   available
Mem:         500144      160208        6548        5168      333388      307548
Swap:       1997820           0     1997820
```

[stextbox id='black' image='null'] **Important**: If the file does not exist, it is automatically created. [/stextbox]

Repeating a single redirection will replace the file:

```
trainee@ubuntu1604:~/training$ date > file
trainee@ubuntu1604:~/training$ cat file
Mon 28 Nov 15:51:07 CET 2016
```

To add additional data to the file, you need to use a **double redirection**:

```
trainee@ubuntu1604:~/training$ free >> file
trainee@ubuntu1604:~/training$ cat file
Mon 28 Nov 15:51:07 CET 2016
              total        used        free      shared  buff/cache   available
Mem:         500144      160208        6436        5168      333500      307548
Swap:       1997820           0     1997820
```

[stextbox id='black' image='null'] **Important** : Note that standard output can only be redirected to a single destination. [/stextbox]

File descriptors are numbered for ease of use :

- 0 = Standard Input
- 1 = Standard Output
- 2 = Standard Error

For example:

```
trainee@ubuntu1604:~/training$ cd ..
trainee@ubuntu1604:~$ rmdir training/ 2>errorlog
trainee@ubuntu1604:~$ cat erreurlog
rmdir: failed to remove 'training/': Directory not empty
```

[stextbox id='black' image='null'] **Important**: As you can see the error generated is redirected to the **errorlog** file. [/stextbox]

You can join file descriptors using the **&** character:

```
trainee@ubuntu1604:~$ free > file 2>&1
```

Any errors are sent to the same destination as the standard output, in the case, **file**.

It is also possible to have a reverse redirection:

```
trainee@ubuntu1604:~$ wc -w < errorlog
8
```

In this case **wc -w** counts the number of words in the file.

Other redirections exist :

Redirection	**Definition**
&>	Join file descriptors 1 and 2.
<<	Takes the text typed on the next lines as standard input until EOF is found at the beginning of a line.
<>	Allows the use of the same file as STDIN and STDOUT.

Pipes

A pipe is used to present the standard output on the first command to the standard input of the second command:

```
trainee@ubuntu1604:~$ ls | wc -w
17
```

[stextbox id='black' image='null'] **Important** - Several pipes can be used within the same command. [/stextbox]

Standard output can generally only be redirected to a single destination. To redirect to two destinations at once, you need to use the **tee** command:

```
trainee@ubuntu1604:~$ date | tee file1
Mon 28 Nov 16:14:22 CET 2016
trainee@ubuntu1604:~$ cat file1
Mon 28 Nov 16:14:22 CET 2016
```

Alternatively, tee can be used to redirect to two files at the same time:

```
trainee@ubuntu1604:~$ date | tee file1 > file2
trainee@ubuntu1604:~$ cat file1
Mon 28 Nov 16:15:56 CET 2016
trainee@ubuntu1604:~$ cat file2
Mon 28 Nov 16:15:56 CET 2016
```

[stextbox id='black' image='null'] **Important** : The default action of the **tee** command is to overwrite the destination file. In order to append output to the same file, you need to use the **-a** switch. [/stextbox]

Command Substitution

Command substitution permits in-line execution of a command:

```
trainee@ubuntu1604:~$ echo date
date
trainee@ubuntu1604:~$ echo $(date)
Mon 28 Nov 16:19:33 CET 2016
trainee@ubuntu1604:~$ echo `date`
Mon 28 Nov 16:19:33 CET 2016
```

Conditional Command Execution

Commands can be grouped using brackets:

```
$ (ls -l; ps; who) > list
```

Conditional command execution can be obtained by using the exit status value and either **&&** or **||**.

For example,

- Command1 && Command2,
    - Command2 will execute if the exit status of Command1 is 0,
- Command1 || Command2,
    - Command2 will execute if the exit status of Command1 anything other than 0.

Environment Variables

The contents of a shell variable can be displayed on standard output using the **echo** command:

```
$ echo $VARIABLE [Enter]
```

Principal Variables

Variable	**Description**
BASH	Complete path to current shell.
BASH_VERSION	Shell version.
EUID	EUID of the current user.
UID	UID of the current user.
PPID	PID of the parent of the current process.
PWD	The current directory.
OLDPWD	The previous current directory (like the **cd -**command).
RANDOM	A random number between 0 and 32767.
SECONDS	The numbers of seconds since the shell was started.
LINES	The number of lines in a screen.
COLUMNS	The number of columns in a screen .
HISTFILE	The history file.
HISTFILESIZE	The history file size.
HISTSIZE	The number of commands that can be saved to the history file.
HISTCMD	The current command's number in the History.
HISTCONTROL	**ignorespace** or **ignoredups** or **ignoreboth**

Variable	Description
HOME	The user's home directory.
HOSTTYPE	Machine type.
OSTYPE	The OS type.
MAIL	The file containing the user's mail.
MAILCHECK	Frequency in seconds that a user's mail is checked.
PATH	The paths to executables.
PROMPT_COMMAND	Command executed before each prompt is displayed.
PS1	User's default prompt.
PS2	User's 2nd level default prompt.
PS3	User's 3rd level prompt.
PS4	User's 4th level prompt.
SHELL	User's current shell.
SHLVL	The number of shell instances.
TMOUT	The number of seconds less 60 before an unused terminal gets sent the **exit** command.

Internationalisation and Localisation

Internationalisation, also called **i18n** since there are 18 letters between the **I** and **n**, consists of modifying software so that it conforms to regional parameters:

- Text processing differences,
- Writing direction,
- Different systems of numerals,
- Telephone numbers, addresses and international postal codes,
- Weights and measures,
- Date/time format,
- Paper sizes,
- Keyboard layout,
- etc ...

Localisation, also called **L10n** since there are 10 letters between the **L** and **n**, consists of modifying the Internationalisation so that it conforms to a

specific locale:

- en_GB = Great Britain,
- en_US = USA,
- en_AU = Australia,
- en_NZ = New Zealand,
- en_ZA = South Africa,
- en_CA = Canada.

The most important variables are:

```
trainee@ubuntu1604:~$ echo $LC_ALL
en_GB.UTF-8
trainee@ubuntu1604:~$ echo $LC_CTYPE

trainee@ubuntu1604:~$ echo $LANG
en_GB.UTF-8

trainee@ubuntu1604:~$ locale
LANG=en_GB.UTF-8
LANGUAGE=
LC_CTYPE="en_GB.UTF-8"
LC_NUMERIC="en_GB.UTF-8"
LC_TIME="en_GB.UTF-8"
LC_COLLATE="en_GB.UTF-8"
LC_MONETARY="en_GB.UTF-8"
LC_MESSAGES="en_GB.UTF-8"
LC_PAPER="en_GB.UTF-8"
LC_NAME="en_GB.UTF-8"
LC_ADDRESS="en_GB.UTF-8"
LC_TELEPHONE="en_GB.UTF-8"
LC_MEASUREMENT="en_GB.UTF-8"
LC_IDENTIFICATION="en_GB.UTF-8"
LC_ALL=en_GB.UTF-8
```

Special Variables

Variable	Description
$LINENO	Contains the current line number of the script or function being executed
$$	Contains the PID of the current process
$PPID	Contains the PID of the parent of the current process
$0	Contains the name of the current script
$1, $2 ...	Contains respectively the 1st, 2nd etc arguments passed to the script
$#	Contains the total number of arguments passed to the script
$*	Contains all of the arguments passed to the script
$@	Contains all of the arguments passed to the script

The env Commande

The **env** command can be used to run a program in a modified environment or just list the values of all environmental variables associated with the user calling the program env:

```
trainee@ubuntu1604:~$ env
LC_PAPER=fr_FR.UTF-8
LC_ADDRESS=fr_FR.UTF-8
XDG_SESSION_ID=1
LC_MONETARY=fr_FR.UTF-8
TERM=xterm-256color
SHELL=/bin/bash
SSH_CLIENT=10.0.2.2 40266 22
LC_NUMERIC=fr_FR.UTF-8
OLDPWD=/home/trainee/training
SSH_TTY=/dev/pts/8
LC_ALL=en_GB.UTF-8
USER=trainee
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37
;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=0
1;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31
```

```
:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*
.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=
01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=01;35
:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.ti
ff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=
01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;3
5:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl
=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36
:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.w
av=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
LC_TELEPHONE=fr_FR.UTF-8
MAIL=/var/mail/trainee
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
QT_QPA_PLATFORMTHEME=appmenu-qt5
LC_IDENTIFICATION=fr_FR.UTF-8
PWD=/home/trainee
LANG=en_US.UTF-8
LC_MEASUREMENT=fr_FR.UTF-8
SHLVL=1
HOME=/home/trainee
LOGNAME=trainee
SSH_CONNECTION=10.0.2.2 40266 10.0.2.15 22
LESSOPEN=| /usr/bin/lesspipe %s
XDG_RUNTIME_DIR=/run/user/1000
LESSCLOSE=/usr/bin/lesspipe %s %s
LC_TIME=fr_FR.UTF-8
LC_NAME=fr_FR.UTF-8
_=/usr/bin/env
```

To run a program, such as **xterm** in a modified environment the command is:

```
$ env EDITOR=vim xterm
```

Bash Shell Options

To view all the options of the bash shell, use the command **set**:

```
trainee@ubuntu1604:~$ set -o
allexport       off
braceexpand     on
emacs           on
errexit         off
errtrace        off
functrace       off
hashall         on
histexpand      on
history         on
ignoreeof       off
interactive-comments    on
keyword         off
monitor         on
noclobber       off
noexec          off
noglob          off
nolog           off
notify          off
nounset         off
onecmd          off
physical        off
pipefail        off
posix           off
privileged      off
verbose         off
vi              off
xtrace          off
```

To turn on an option you need to specify which option as an argument to the previous command:

```
trainee@ubuntu1604:~$ set -o allexport
trainee@ubuntu1604:~$ set -o
allexport       on
braceexpand     on
...
```

To turn off an option, use set with the **+o** option:

```
trainee@ubuntu1604:~$ set +o allexport
trainee@ubuntu1604:~$ set -o
allexport       off
braceexpand     on
...
```

These are the most interesting options:

Option	Default value	Description
allexport	off	The shell automatically exports all variables
emacs	on	emacs editing mode
noclobber	off	Simple re-directions do not squash the target file if it exists
noglob	off	Turns off special characters
nounset	off	The shell will return an error if the variable is not set
verbose	off	Echos back the typed command
vi	off	vi editing mode

noclobber

```
trainee@ubuntu1604:~$ set -o noclobber
trainee@ubuntu1604:~$ pwd > file
trainee@ubuntu1604:~$ pwd > file
-bash: file: cannot overwrite existing file
```

```
trainee@ubuntu1604:~$ pwd >| file
trainee@ubuntu1604:~$ set +o noclobber
```

[stextbox id='black' image='null'] **Important** : Note that the **noclobber** option can be overidden by using a pipe. [/stextbox]

noglob

```
trainee@ubuntu1604:~$ set -o noglob
trainee@ubuntu1604:~$ echo *
*
trainee@ubuntu1604:~$ set +o noglob
trainee@ubuntu1604:~$ echo *
aac abc bca codes Desktop Documents Downloads errorlog examples.desktop file file1 Music Pictures Public
Templates training Videos vitext xyz
```

[stextbox id='black' image='null'] **Important** : Note that metacharacters are turned off when the **noglob** option is set. [/stextbox]

nounset

```
trainee@ubuntu1604:~$ set -o nounset
trainee@ubuntu1604:~$ echo $FENESTROS
-bash: FENESTROS: unbound variable
trainee@ubuntu1604:~$ set +o nounset
trainee@ubuntu1604:~$ echo $FENESTROS

trainee@ubuntu1604:~$
```

[stextbox id='black' image='null'] **Important** : Note that the inexistant variable **$FENESTROS** is identified as such when the **nounset** option is set. [/stextbox]

Basic Shell Scripting

Execution

A script is a text file that is read by the system and it's contents executed. There are five ways to execute a script:

By stipulating the shell that will execute the script:

/bin/bash myscript

by a reverse redirection:

/bin/bash < myscript

By calling the script by it's name, provided that the script is executable and that it resides in a directory specified by your path :

myscript

By placing yourself in the directory where the script resides and using one of the two following possibilities :

. myscript et **./myscript**

[stextbox id='black' image='null'] **Important**: In the first case the script is executed in the parent shell. In the second case the script is executed in a child shell. [/stextbox]

Comments in a script are lines starting with **#**. However, each script starts with a pseudo-comment that informs the system which shell should be used to execute the script:

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

Since a script in it's simplest form is a list of commands that are sequentially executed, it is often useful to test those command prior to writing the script> Linux has a command that can help you debug a future script. The **script** command can be used to generate a log file, called **typescript**, that contains a record of everything occurred on standard output. To exit the recording mode, use **exit**:

```
trainee@ubuntu1604:~$ script
Script started, file is typescript
trainee@ubuntu1604:~$ pwd
/home/trainee
trainee@ubuntu1604:~$ ls
aac  codes      Downloads         file1  file1     Public     typescript  xyz
abc  Desktop    errorlog          file2  Music     Templates  Videos
bca  Documents  examples.desktop  file      Pictures  training   vitext
trainee@ubuntu1604:~$ exit
exit
Script done, file is typescript
trainee@ubuntu1604:~$ cat typescript
Script started on Tue 29 Nov 2016 03:57:47 CET
trainee@ubuntu1604:~$ pwd
/home/trainee
trainee@ubuntu1604:~$ ls
aac  codes      Downloads         file1  file1     Public     typescript  xyz
abc  Desktop    errorlog          file2  Music     Templates  Videos
bca  Documents  examples.desktop  file      Pictures  training   vitext
trainee@ubuntu1604:~$ exit
exit

Script done on Tue 29 Nov 2016 03:57:58 CET
```

Lets start by creating a simple script called **myscript**:

```
$ vi myscript [Enter]
```

Edit the file as follows:

```
pwd
ls
```

[stextbox id='black' image='null'] **Important**: Note that in the above example, the script does not start with a pseudo-comment. As a result, the script will be executed by the shell of the user that invokes it unless a different shell is specified. [/stextbox]

Save the file and use the five ways to execute it.

As an argument de /bin/bash:

```
trainee@ubuntu1604:~$ vi myscript
trainee@ubuntu1604:~$ /bin/bash myscript
/home/trainee
aac  codes  Downloads      file1    Pictures   training    vitext
abc  Desktop    errorlog      file2  myscript  Public      typescript  xyz
bca  Documents  examples.desktop  file        Music      Templates  Videos
```

Using a redirection:

«code> trainee@ubuntu1604:~$ /bin/bash < myscript /home/trainee aac codes Downloads file1 Pictures training vitext abc Desktop errorlog file2 myscript Public typescript xyz bca Documents examples.desktop file Music Templates Videos </code>

In order to be able to call the script by it's name from another directory, such as **/tmp**, you need to move the script into the **/home/trainee/bin** directory and make it executable. Note that in this case, the the value of the environmental variable $PATH should contain a reference to **/home/trainee/bin**:

```
trainee@ubuntu1604:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
```

As you can see, in the case of Ubuntu, this is not the case. The reason for this becomes apparent when viewing the contents of the **.profile** file in **/home/trainee**:

```
trainee@ubuntu1604:~$ cat .profile
# ~/.profile: executed by the command interpreter for login shells.
...
# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
```

```
    PATH="$HOME/bin:$PATH"
fi
```

As you can see PATH is set so it includes the user's private bin only if the directory exists:

```
# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
```

To fix the value of the PATH variable, create the $HOME/bin directory and re-load the .profile file:

«code> trainee@ubuntu1604:~$ mkdir bin trainee@ubuntu1604:~$ source .profile trainee@ubuntu1604:~$ echo $PATH
/home/trainee/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin </code>

Now you need to move the script to $HOME/bin and make it executable:

```
trainee@ubuntu1604:~$ mv myscript ~/bin
trainee@ubuntu1604:~$ chmod u+x ~/bin/myscript
```

Move to **/tmp** and can call the script by just using it's name:

```
trainee@ubuntu1604:/tmp$ myscript
/tmp
hsperfdata_root
systemd-private-2596faf2be00473d9dc6da53af5711d5-colord.service-K4xRp2
systemd-private-2596faf2be00473d9dc6da53af5711d5-rtkit-daemon.service-iKio6G
systemd-private-2596faf2be00473d9dc6da53af5711d5-systemd-timesyncd.service-AlOq91
```

Now move back to ~/bin and use the following two commands to execute myscript:

- ./myscript
- . myscript

```
trainee@ubuntu1604:/tmp$ cd ~/bin
trainee@ubuntu1604:~/bin$ ./myscript
/home/trainee/bin
myscript
trainee@ubuntu1604:~/bin$ . myscript
/home/trainee/bin
myscript
```

[stextbox id='black' image='null'] **To do**: Note the difference in the output of these two commands and explain that difference. [/stextbox]

The read command

HERE

The read command reads the standard input and stores the information in the variables that are specified as arguments. The separator between fields is a space, a tabultaion or a carriage return:

```
trainee@ubuntu1604:~/bin$ read var1 var2 var3 var4
fenestros edu is great!
trainee@ubuntu1604:~/bin$ echo $var1
fenestros
trainee@ubuntu1604:~/bin$ echo $var2
edu
trainee@ubuntu1604:~/bin$ echo $var3
is
trainee@ubuntu1604:~/bin$ echo $var4
great!
```

[stextbox id='black' image='null'] **Important**: Note that each field has been placed in a seperate variable. Note also that by convention, user declared variables are in lower case in order to distinguish them from system variables. [/stextbox]

```
trainee@ubuntu1604:~/bin$ read var1 var2
```

```
fenestros edu is great!
trainee@ubuntu1604:~/bin$ echo $var1
fenestros
trainee@ubuntu1604:~/bin$ echo $var2
edu is great!
```

[stextbox id='black' image='null'] **Important**: Note that in this case, $var2 contains three fields. [/stextbox]

Code de retour

The contents of a variable can also be empty:

```
trainee@ubuntu1604:~/bin$ read var
```

↵ Enter

```
trainee@ubuntu1604:~/bin$ echo $?
0
trainee@ubuntu1604:~/bin$ echo $var

trainee@ubuntu1604:~/bin$
```

But not null:

```
trainee@ubuntu1604:~/bin$ read var
```

Ctrl+D

```
trainee@ubuntu1604:~/bin$ echo $?
1
trainee@ubuntu1604:~/bin$ echo $var
```

```
trainee@ubuntu1604:~/bin$
```

The IFS Variable

The IFS variable contains the default separator characters: SpaceBar, Tab ⇆ and ↵ Enter:

```
trainee@ubuntu1604:~/bin$ echo "$IFS" | od -c
0000000      \t  \n  \n
0000004
```

[stextbox id='black' image='null'] **Important**: The **od** command (*Octal Dump*) returns the contents of a file in octal format. The **-c** switch prints to standard output any ASCII characters or backslashes contained within the file. [/stextbox]

It is possible to change the contents of this variable:

```
trainee@ubuntu1604:~/bin$ OLDIFS="$IFS"
trainee@ubuntu1604:~/bin$ IFS=":"
trainee@ubuntu1604:~/bin$ echo "$IFS" | od -c
0000000   :  \n
0000002
```

Now test the new configuration:

```
trainee@ubuntu1604:~/bin$ read var1 var2 var3
fenestros:edu is:great!
trainee@ubuntu1604:~/bin$ echo $var1
fenestros
trainee@ubuntu1604:~/bin$ echo $var2
edu is
trainee@ubuntu1604:~/bin$ echo $var3
great!
```

Restore the old value of IFS before proceeding further: IFS="$OLDIFS"

```
trainee@ubuntu1604:~/bin$ IFS="$OLDIFS"
trainee@ubuntu1604:~/bin$ echo "$IFS" | od -c
0000000      \t  \n  \n
0000004
```

The test Command

The **test** command uses two forms:

test *expression*

or

[SpaceBar*expression*SpaceBar**]**

Testing Files

Test	Description
-f file	Returns true if file is an ordinary file
-d file	Returns true if file is a directory
-r file	Returns true if user can read file
-w file	Returns true if user can write file
-x file	Returns true if user can execute file
-e file	Returns true if file exists
-s file	Returns true if file is not empty
file1 -nt file2	Returns true if file1 is newer than file2
file1 -ot file2	Returns true if file1 is older than file2
file1 -ef file2	Returns true if file1 is identical to file2

LAB #1

Test whether the **a100** file is an ordinary file:

```
trainee@ubuntu1604:~/bin$ cd ../training/
trainee@ubuntu1604:~/training$ test -f a100
trainee@ubuntu1604:~/training$ echo $?
0
trainee@ubuntu1604:~/training$ [ -f a100 ]
trainee@ubuntu1604:~/training$ echo $?
0
```

[stextbox id='black' image='null'] **Important**: The value contained in $? is 0. This indicates **true**. [/stextbox]

Test whether the **a101** file is an ordinary file:

```
trainee@ubuntu1604:~/training$ [ -f a101 ]
trainee@ubuntu1604:~/training$ echo $?
1
```

[stextbox id='black' image='null'] **Important**: The value contained in $? is 1. This indicates **false**. This is obvious since a101 does not exist. [/stextbox]

Test whether **/home/trainee/training** is a directory:

```
trainee@ubuntu1604:~/training$ [ -d /home/trainee/training ]
trainee@ubuntu1604:~/training$ echo $?
0
```

[stextbox id='black' image='null'] **Important**: The value contained in $? is 0. This indicates **true**. [/stextbox]

Testing Strings

Test	Description
-n string	Returns true if string is not zero in length
-z string	Returns true if string is zero in length
string1 = string2	Returns true if string1 is equal to string2
string1 != string2	Returns true if string1 is different to string2
string1	Returns true if string1 is not empty

LAB #2

Test whether two strings are indentical:

```
trainee@ubuntu1604:~/training$ string1="root"
trainee@ubuntu1604:~/training$ string2="fenestros"
trainee@ubuntu1604:~/training$ [ $string1 = $string2 ]
trainee@ubuntu1604:~/training$ echo $?
1
```

[stextbox id='black' image='null'] **Important**: The value contained in $? is 1. This indicates **false**. [/stextbox]

Test if string1 is not zero in length:

```
trainee@ubuntu1604:~/training$ [ -n $string1 ]
trainee@ubuntu1604:~/training$ echo $?
0
```

[stextbox id='black' image='null'] **Important**: The value contained in $? is 0. This indicates **true**. [/stextbox]

Test if string1 is is zero in length:

```
trainee@ubuntu1604:~/training$ [ -z $string1 ]
trainee@ubuntu1604:~/training$ echo $?
1
```

[stextbox id='black' image='null'] **Important**: The value contained in $? is 1. This indicates **false**. [/stextbox]

Testing Numbers

Test	Description
value1 -eq value2	Returns true if value1 is equal to value2
value1 -ne value2	Returns true if value1 is not equal to value2
value1 -lt value2	Returns true if value1 is less than value2
value1 -le value2	Returns true if value1 is less than or equal to value2
value1 -gt value2	Returns true if value1 is greater than value2
value1 -ge value2	Returns true if value1 is greater than or equal to value2

LAB #3

Compare the two numbers **value1** and **value2** :

```
trainee@ubuntu1604:~/training$ read value1
35
trainee@ubuntu1604:~/training$ read value2
23
trainee@ubuntu1604:~/training$ [ $value1 -lt $value2 ]
trainee@ubuntu1604:~/training$ echo $?
1
trainee@ubuntu1604:~/training$ [ $value2 -lt $value1 ]
trainee@ubuntu1604:~/training$ echo $?
0
trainee@ubuntu1604:~/training$ [ $value2 -eq $value1 ]
trainee@ubuntu1604:~/training$ echo $?
1
```

Expressions

Test	Description
!expression	Returns true if expression is false
expression1 -a expression2	Represents a logical OR between expression1 and expression2
expression1 -o expression2	Represents a logical AND between expression1 and expression2
\(expression\)	Parenthesis let you group together expressions

LAB #4

Test if $file is not a directory:

```
trainee@ubuntu1604:~/training$ file=a100
trainee@ubuntu1604:~/training$ [ ! -d $file ]
trainee@ubuntu1604:~/training$ echo $?
0
```

Test if $directory is a directory and if trainee can cd into it:

```
trainee@ubuntu1604:~/training$ directory=/usr
trainee@ubuntu1604:~/training$ [ -d $directory -a -x $directory ]
trainee@ubuntu1604:~/training$ echo $?
0
```

Test if trainee has the write permission for the a100 file **and** test if /usr is a directory **or** test if /tmp is a directory:

```
trainee@ubuntu1604:~/training$ [ -w a100 -a \( -d /usr -o -d /tmp \) ]
trainee@ubuntu1604:~/training$ echo $?
0
```

Testing the User Environment

Test	Description
-o option	Returns true if the shell option “option” is on

LAB #5

```
trainee@ubuntu1604:~/training$ [ -o allexport ]
trainee@ubuntu1604:~/training$ echo $?
1
```

The [[expression]] Command

The **[[SpaceBarexpressionSpaceBar]]** command is an improved **test** command with some minor changes to syntax:

Test	Description
expression1 && expression2	Represents a logical OR between expression1 and expression2
expression1 || expression2	Represents a logical AND between expression1 and expression2
(expression)	Parenthesis let you group together expressions

and some additional operators :

Test	Description
string = model	Returns true if string corresponds to model
string != model	Returns true if string does not correspond to model
string1 < string2	Returns true if string1 is lexicographically before string2
string1 > string2	Returns true if string1 is lexicographically after string2

LAB #6

Test if trainee has the write permission for the a100 file **and** test if /usr is a directory **or** test if /tmp is a directory:

```
trainee@ubuntu1604:~/training$ [[ -w a100 && ( -d /usr || -d /tmp ) ]]
trainee@ubuntu1604:~/training$ echo $?
0
```

Shell Operators

Operator	Description
Command1 && Command2	Command2 is executed if the exit code of Command1 is zero
Command1 || Command2	Command2 is executed is the exit code of Command1 is not zero

LAB #7

```
trainee@ubuntu1604:~/training$ [[ -d /root ]] && echo "The root directory exists"
The root directory exists
trainee@ubuntu1604:~/training$ [[ -d /root ]] || echo "The root directory exists"
trainee@ubuntu1604:~/training$
```

The expr Command

The**expr** command's syntax is as follows :

expr SpaceBar number1 SpaceBar *operator* SpaceBar number2 SpaceBar

ou

expr Tab ⇆ number1 Tab *operator* Tab ⇆ number2 ↵ Enter

ou

expr SpaceBar string SpaceBar : SpaceBar *regular_expression* SpaceBar

or

expr Tab ⇆ string Tab ⇆ : Tab ⇆ *regular_expression* ↵ Enter

Maths

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo
\(\)	Parentheses

Comparisons

Operator	Description
\<	Less than
\<=	Less than or equal to
\>	Greater then
\>=	Greater then or equal to
=	Equal to
!=	Not equal to

Logic

Operator	Description
|	Logical OR
\&	Logical AND

LAB #8

Add two to the value of $x:

```
trainee@ubuntu1604:~/training$ x=2
trainee@ubuntu1604:~/training$ expr $x + 2
4
```

If the surrounding spaces are removed, the result is completely different:

```
trainee@ubuntu1604:~/training$ expr $x+2
2+2
```

Certain operators need to be protected:

```
trainee@ubuntu1604:~/training$ expr $x * 2
expr: syntax error
trainee@ubuntu1604:~/training$ expr $x \* 2
4
```

Now put the result of a calculation in a variable:

```
trainee@ubuntu1604:~/training$ resultat=`expr $x + 10`
trainee@ubuntu1604:~/training$ echo $resultat
12
```

The let Command

The let command is equivalent to ((expression)). The ((expression)) command provides the following additional features when compared with the **expr** command :

- greater number of operators,
- no need for spaces or tabulations between arguments,
- no need to prefix variables with the **$** character,
- the shell's special characters do not need to be escaped,
- variables are defined directly in the command,

- faster execution time.

Maths

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo
^	Power

Comparisons

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater then
>=	Greater then or equal to
==	Equal
!=	Not Equal

Logic

Operator	Description
&&	Logical AND
||	Logical OR
!	Logical negation

Binary

Opérateur	Description
~	Binary negation
>>	décalage binaire à droite
<<	décalage binaire à gauche
&	Binary AND
|	Binary OR
^	Exclusive binary OR

LAB #9

```
trainee@ubuntu1604:~/training$ x=2
trainee@ubuntu1604:~/training$ ((x=$x+10))
trainee@ubuntu1604:~/training$ echo $x
12
trainee@ubuntu1604:~/training$ ((x=$x+20))
trainee@ubuntu1604:~/training$ echo $x
32
```

Control Structures

If

The syntax is as follows:

```
if condition
then
    command(s)
else
    command(s)
fi
```

or:

```
if condition
then
    command(s)
    command(s)
fi
```

or finally:

```
if condition
then
     command(s)
elif condition
then
     command(s)
elif condition
then
     command(s)
else
    command(s)

fi
```

case

The syntax is as follows:

```
case $variable in
model1) function
   ...
   ;;
```

```
model2) function
   ...
   ;;
model3 | model4 | model5 ) function
   ...
   ;;
esac
```

Loops

for

The syntax is as follows:

```
for variable in variable_list
do
    command(s)
done
```

while

The syntax is as follows:

```
while condition
do
    command(s)
done
```

Example

```
U=1
while [ $U -lt $MAX_ACCOUNTS ]
do
useradd fenestros"$U" -c fenestros"$U" -d /home/fenestros"$U" -g staff -G audio,fuse -s /bin/bash 2>/dev/null
useradd fenestros"$U"$ -g machines -s /dev/false -d /dev/null 2>/dev/null
echo "Compte fenestros$U créé"
let U=U+1
done
```

Start-up Scripts

When Bash is called as a login shell it executes the start-up scripts in the following order:

- **/etc/profile**,
- **~/.bash_profile** or **~/.bash_login** or **~/.profile** dependant upon the distribution,

In the cas of Ubuntu Bash executes **~/.profile**.

When a login shell is terminated, Bash executes the **~/.bash_logout** file if it exists.

Whan Bash is called as an interactive shell as opposed to a login shell, it executes only the **~/.bashrc** file.

LAB #10

[stextbox id='black' image='null'] **To do** : Using the knowledge you have acquired in this unit, explain each of the following scripts. [/stextbox]

~/.profile

```
trainee@ubuntu1604:~/training$ cat ~/.profile
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
    . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
```

~/.bashrc

```
trainee@ubuntu1604:~/training$ cat ~/.bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
```

```
case $- in
    *i*) ;;
      *) return;;
esac

# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize

# If set, the pattern "**" used in a pathname expansion context will
# match all files and zero or more directories and subdirectories.
#shopt -s globstar

# make less more friendly for non-text input files, see lesspipe(1)
[ -x /usr/bin/lesspipe ] && eval "$(SHELL=/bin/sh lesspipe)"

# set variable identifying the chroot you work in (used in the prompt below)
if [ -z "${debian_chroot:-}" ] && [ -r /etc/debian_chroot ]; then
    debian_chroot=$(cat /etc/debian_chroot)
fi

# set a fancy prompt (non-color, unless we know we "want" color)
case "$TERM" in
```

```
    xterm-color|*-256color) color_prompt=yes;;
esac

# uncomment for a colored prompt, if the terminal has the capability; turned
# off by default to not distract the user: the focus in a terminal window
# should be on the output of commands, not on the prompt
#force_color_prompt=yes

if [ -n "$force_color_prompt" ]; then
    if [ -x /usr/bin/tput ] && tput setaf 1 >&/dev/null; then
    # We have color support; assume it's compliant with Ecma-48
    # (ISO/IEC-6429). (Lack of such support is extremely rare, and such
    # a case would tend to support setf rather than setaf.)
    color_prompt=yes
    else
    color_prompt=
    fi
fi

if [ "$color_prompt" = yes ]; then
    PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ '
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
fi
unset color_prompt force_color_prompt

# If this is an xterm set the title to user@host:dir
case "$TERM" in
xterm*|rxvt*)
    PS1="\[\e]0;${debian_chroot:+($debian_chroot)}\u@\h: \w\a\]$PS1"
    ;;
*)
    ;;
esac
```

```
# enable color support of ls and also add handy aliases
if [ -x /usr/bin/dircolors ]; then
    test -r ~/.dircolors && eval "$(dircolors -b ~/.dircolors)" || eval "$(dircolors -b)"
    alias ls='ls --color=auto'
    #alias dir='dir --color=auto'
    #alias vdir='vdir --color=auto'

    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
    alias egrep='egrep --color=auto'
fi

# colored GCC warnings and errors
#export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:locus=01:quote=01'

# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands.  Use like so:
#   sleep 10; alert
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)" "$(history|tail -n1|sed
-e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')"'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi
```

```
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
```

<html>

</html>