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 SLES 12 **/bin/sh** is a soft link to **/bin/bash** :

```
trainee@SLES12SP1:~> ls -l /bin/sh
lrwxrwxrwx 1 root root 4  1 mai    2016 /bin/sh -> bash
```

/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@SLES12SP1:~> 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@SLES12SP1:~> 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@SLES12SP1:~> type ls
ls is aliased to `_ls'
```

[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@SLES12SP1:~> alias dir='ls -l'
trainee@SLES12SP1:~> dir
total 4
-rw-r--r-- 1 trainee users   0  1 oct.  06:55 aac
-rw-r--r-- 1 trainee users   0  1 oct.  06:55 abc
-rw-r--r-- 1 trainee users   0  1 oct.  06:55 bca
drwxr-xr-x 1 trainee users   0  1 mai    2016 bin
drwxr-xr-x 1 trainee users   0  2 mai    2016 Desktop
drwxr-xr-x 1 trainee users   0  2 mai    2016 Documents
drwxr-xr-x 1 trainee users   0  2 mai    2016 Downloads
drwxr-xr-x 1 trainee users   0  2 mai    2016 Music
drwxr-xr-x 1 trainee users   0  2 mai    2016 Pictures
drwxr-xr-x 1 trainee users   0  2 mai    2016 Public
drwxr-xr-x 1 trainee users  20  1 mai    2016 public_html
drwxr-xr-x 1 trainee users   0  2 mai    2016 Templates
drwxr-xr-x 1 trainee users   0  2 mai    2016 Videos
-rw-r--r-- 1 trainee users 391 30 sept. 10:27 vitext
-rw-r--r-- 1 trainee users   0  1 oct.  06:55 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@SLES12SP1:~> alias
alias +='pushd .'
alias -='popd'
alias ..='cd ..'
alias ...='cd ../..'
alias aumix='padsp aumix'
alias beep='echo -en "\007"'
alias cd..='cd ..'
alias dir='ls -l'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -alF'
alias la='ls -la'
alias ll='ls -l'
alias ls='_ls'
alias ls-l='ls -l'
alias md='mkdir -p'
alias o='less'
alias rd='rmdir'
alias rehash='hash -r'
alias sox='padsp sox'
alias timidity='timidity -Oe'
alias unmount='echo "Error: Try the command: umount" 1>&2; false'
alias you='if test "$EUID" = 0 ; then /sbin/yast2 online_update ; else su - -c "/sbin/yast2 online_update" ; fi'
```

[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@SLES12SP1:~> \dir
aac  bca  Desktop    Downloads  Pictures  public_html  Videos  xyz
abc  bin  Documents  Music  Public      Templates    vitext
```

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

```
trainee@SLES12SP1:~> unalias dir
trainee@SLES12SP1:~> dir
aac  bca  Desktop    Downloads  Pictures  public_html  Videos  xyz
abc  bin  Documents  Music  Public      Templates    vitext
```

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

```
trainee@SLES12SP1:~> cat /etc/passwd
at:x:25:25:Batch jobs daemon:/var/spool/atjobs:/bin/bash
bin:x:1:1:bin:/bin:/bin/bash
daemon:x:2:2:Daemon:/sbin:/bin/bash
ftp:x:40:49:FTP account:/srv/ftp:/bin/bash
ftpsecure:x:488:65534:Secure FTP User:/var/lib/empty:/bin/false
games:x:12:100:Games account:/var/games:/bin/bash
gdm:x:486:485:Gnome Display Manager daemon:/var/lib/gdm:/bin/false
lp:x:4:7:Printing daemon:/var/spool/lpd:/bin/bash
mail:x:8:12:Mailer daemon:/var/spool/clientmqueue:/bin/false
man:x:13:62:Manual pages viewer:/var/cache/man:/bin/bash
messagebus:x:499:499:User for D-Bus:/var/run/dbus:/bin/false
news:x:9:13:News system:/etc/news:/bin/bash
nobody:x:65534:65533:nobody:/var/lib/nobody:/bin/bash
nscd:x:496:495:User for nscd:/run/nscd:/sbin/nologin
ntp:x:74:492:NTP daemon:/var/lib/ntp:/bin/false
openslp:x:494:2:openslp daemon:/var/lib/empty:/sbin/nologin
polkitd:x:497:496:User for polkitd:/var/lib/polkit:/sbin/nologin
postfix:x:51:51:Postfix Daemon:/var/spool/postfix:/bin/false
```

```
pulse:x:490:489:PulseAudio daemon:/var/lib/pulseaudio:/sbin/nologin
root:x:0:0:root:/root:/bin/bash
rpc:x:495:65534:user for rpcbind:/var/lib/empty:/sbin/nologin
rtkit:x:491:490:RealtimeKit:/proc:/bin/false
scard:x:487:487:Smart Card Reader:/var/run/pcscd:/usr/sbin/nologin
sshd:x:498:498:SSH daemon:/var/lib/sshd:/bin/false
statd:x:489:65534:NFS statd daemon:/var/lib/nfs:/sbin/nologin
usbmux:x:493:65534:usbmuxd daemon:/var/lib/usbmuxd:/sbin/nologin
uucp:x:10:14:Unix-to-Unix CoPy system:/etc/uucp:/bin/bash
vnc:x:492:491:user for VNC:/var/lib/empty:/sbin/nologin
wwwrun:x:30:8:WWW daemon apache:/var/lib/wwwrun:/bin/false
trainee:x:1000:100:trainee:/home/trainee:/bin/bas
```

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

```
trainee@SLES12SP1:~> cat /etc/shells
/bin/ash
/bin/bash
/bin/csh
/bin/dash
/bin/false
/bin/ksh
/bin/ksh93
/bin/mksh
/bin/pdksh
/bin/sh
/bin/tcsh
/bin/true
/bin/zsh
/usr/bin/csh
/usr/bin/dash
/usr/bin/ksh
/usr/bin/ksh93
/usr/bin/mksh
```

```
/usr/bin/passwd
/usr/bin/pdksh
/usr/bin/bash
/usr/bin/tcsh
/usr/bin/zsh
```

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

```
trainee@SLES12SP1:~> echo $SHELL
/bin/bash
```

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

```
trainee@SLES12SP1:~> 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@SLES12SP1:~> 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@SLES12SP1:~> cat /etc/passwd | grep trainee
trainee:x:1000:100: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@SLES12SP1:~> 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@SLES12SP1:~> history | more
    1  su -
    2  su -
    3  clear
    4  cd /
    5  ls -l
    6  ls -l /var/run
    7  cd /mnt
    8  ls
    9  cd
   10  mount
   11  mount --help
```

```
   12  cat /etc/fstab
   13  umount --help
   14  dumpe2fs /dev/sda1 | grep -i superbloc
   15  ls -ld /dev/console /dev/initctl /dev/loop0 /etc /etc/passwd
   16  ls -ld /dev/console /dev/initctl /etc /etc/passwd
   17  ls -ldi /dev/console /dev/initctl /etc /etc/passwd
   18  cd /tmp; mkdir inode; cd inode; touch file1; ls -ali
   19  ln file1 file2
   20  ls -ali
   21  ln -s file1 file3
   22  ls -ali
   23  su -
--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@SLES12SP1:~> ls
aac  bca  Desktop    Downloads  Pictures  public_html  Videos  xyz
abc  bin  Documents  Music      Public    Templates    vitext
trainee@SLES12SP1:~> !!
ls
aac  bca  Desktop    Downloads  Pictures  public_html  Videos  xyz
abc  bin  Documents  Music      Public    Templates    vitext
```

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

```
trainee@SLES12SP1:~> !131
ls
aac  bca  Desktop    Downloads  Pictures  public_html  Videos  xyz
abc  bin  Documents  Music      Public    Templates    vitext
```

The environmental variables associated with the history are set system-wide in the **/etc/profile** file:

```
trainee@SLES12SP1:~> cat /etc/profile | grep HISTSIZE
    HISTSIZE=1000
    export HISTSIZE
```

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@SLES12SP1:~> nl .bash_history | more
     1  su -
     2  su -
     3  clear
     4  cd /
     5  ls -l
     6  ls -l /var/run
     7  cd /mnt
     8  ls
     9  cd -
    10  mount
    11  mount --help
    12  cat /etc/fstab
    13  umount --help
    14  dumpe2fs /dev/sda1 | grep -i superbloc
    15  ls -ld /dev/console /dev/initctl /dev/loop0 /etc /etc/passwd
    16  ls -ld /dev/console /dev/initctl /etc /etc/passwd
    17  ls -ldi /dev/console /dev/initctl /etc /etc/passwd
    18  cd /tmp; mkdir inode; cd inode; touch file1; ls -ali
    19  ln file1 file2
    20  ls -ali
    21  ln -s file1 file3
    22  ls -ali
    23  su -
--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@SLES12SP1:~> ls .bash
.bash_history  .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@SLES12SP1:~> mo
modeprint       modsign-verify  mount           mouse-test
modetest        more            mountpoint
```

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 ...

Metacharacter	Description
+(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@SLES12SP1:~> mkdir training
trainee@SLES12SP1:~> cd training
trainee@SLES12SP1:~/training> touch f1 f2 f3 f4 f5
```

The * Metacharacter

Now use the Metacharacter *:

```
trainee@SLES12SP1:~/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@SLES12SP1:~/training> touch f52 f62
```

Now use the Metacharacter **?**:

```
trainee@SLES12SP1:~/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@SLES12SP1:~/training> touch a100
```

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

```
trainee@SLES12SP1:~/training> echo [a-f]*
a100 f1 f2 f3 f4 f5 f52 f62
trainee@SLES12SP1:~/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@SLES12SP1:~/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@SLES12SP1:~/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@SLES12SP1:~/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@SLES12SP1:~/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@SLES12SP1:~/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
```

```
direxpand       off
dirspell        off
dotglob         off
execfail        off
expand_aliases  on
extdebug        off
extglob         on
extquote        on
failglob        off
force_fignore   on
globstar        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@SLES12SP1:~/training> touch f f.txt f123.txt f123123.txt f123123123.txt
```

Execute the following command:

```
trainee@SLES12SP1:~/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@SLES12SP1:~/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@SLES12SP1:~/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@SLES12SP1:~/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@SLES12SP1:~/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@SLES12SP1:~/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@SLES12SP1:~/training> echo \* est un caractère spécial
* est un caractère spécial

trainee@SLES12SP1:~/training> echo "* est un caractère spécial"
* est un caractère spécial

trainee@SLES12SP1:~/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@SLES12SP1:~/training> cd ..
trainee@SLES12SP1:~> mkdir codes
trainee@SLES12SP1:~> echo $?
0
trainee@SLES12SP1:~> touch codes/exit.txt
trainee@SLES12SP1:~> rmdir codes
rmdir: failed to remove ‘codes’: Directory not empty
trainee@SLES12SP1:~> 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@SLES12SP1:~> pwd
/home/trainee
trainee@SLES12SP1:~> cd training
trainee@SLES12SP1:~/training> free > file
trainee@SLES12SP1:~/training> cat file
             total       used       free     shared    buffers     cached
Mem:        394524     386024       8500       5716        452     300420
-/+ buffers/cache:      85152     309372
Swap:      2103292          4    2103288
```

[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@SLES12SP1:~/training> date > file
trainee@SLES12SP1:~/training> cat file
Mon 28 Nov 15:48:29 CET 2016
```

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

```
trainee@SLES12SP1:~/training> free >> file
trainee@SLES12SP1:~/training> cat file
Mon 28 Nov 15:48:29 CET 2016
```

```
             total       used       free     shared    buffers     cached
Mem:        394524     386876       7648       5716        452     300936
-/+ buffers/cache:      85488     309036
Swap:      2103292          4    2103288
```

[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@SLES12SP1:~/training> cd ..
trainee@SLES12SP1:~> rmdir training/ 2>errorlog
trainee@SLES12SP1:~> cat errorlog
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@SLES12SP1:~> 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@SLES12SP1:~> 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@SLES12SP1:~> ls | wc -w
18
```

[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@SLES12SP1:~> date | tee file1
Mon 28 Nov 16:14:43 CET 2016
trainee@SLES12SP1:~> cat file1
Mon 28 Nov 16:14:43 CET 2016
```

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

```
trainee@SLES12SP1:~> date | tee file1 > file2
trainee@SLES12SP1:~> cat file1
Mon 28 Nov 16:16:15 CET 2016
trainee@SLES12SP1:~> cat file2
Mon 28 Nov 16:16:15 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@SLES12SP1:~> echo date
date
trainee@SLES12SP1:~> echo $(date)
Mon 28 Nov 16:19:53 CET 2016
trainee@SLES12SP1:~> echo `date`
Mon 28 Nov 16:19:53 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**
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.

Variable	Description
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@SLES12SP1:~> echo $LC_ALL
en_GB.UTF-8
trainee@SLES12SP1:~> echo $LC_CTYPE

trainee@SLES12SP1:~> echo $LANG
en_GB.UTF-8

trainee@SLES12SP1:~> locale
LANG=en_GB.UTF-8
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

Variable	Description
$*	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@SLES12SP1:~> env
LESSKEY=/etc/lesskey.bin
NNTPSERVER=news
MANPATH=/usr/local/man:/usr/share/man
XDG_SESSION_ID=1
HOSTNAME=SLES12SP1
XKEYSYMDB=/usr/X11R6/lib/X11/XKeysymDB
HOST=SLES12SP1
TERM=xterm-256color
SHELL=/bin/bash
PROFILEREAD=true
HISTSIZE=1000
SSH_CLIENT=10.0.2.2 46258 22
MORE=-sl
SSH_TTY=/dev/pts/0
LC_ALL=en_GB.UTF-8
USER=trainee
LS_COLORS=no=00:fi=00:di=01;34:ln=00;36:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=41;33;01:ex=00;32:*
.cmd=00;32:*.exe=01;32:*.com=01;32:*.bat=01;32:*.btm=01;32:*.dll=01;32:*.tar=00;31:*.tbz=00;31:*.tgz=00;31:*.rpm=
00;31:*.deb=00;31:*.arj=00;31:*.taz=00;31:*.lzh=00;31:*.lzma=00;31:*.zip=00;31:*.zoo=00;31:*.z=00;31:*.Z=00;31:*.
gz=00;31:*.bz2=00;31:*.tb2=00;31:*.tz2=00;31:*.tbz2=00;31:*.xz=00;31:*.avi=01;35:*.bmp=01;35:*.fli=01;35:*.gif=01
;35:*.jpg=01;35:*.jpeg=01;35:*.mng=01;35:*.mov=01;35:*.mpg=01;35:*.pcx=01;35:*.pbm=01;35:*.pgm=01;35:*.png=01;35:
*.ppm=01;35:*.tga=01;35:*.tif=01;35:*.xbm=01;35:*.xpm=01;35:*.dl=01;35:*.gl=01;35:*.wmv=01;35:*.aiff=00;32:*.au=0
0;32:*.mid=00;32:*.mp3=00;32:*.ogg=00;32:*.voc=00;32:*.wav=00;32:
```

```
XNLSPATH=/usr/share/X11/nls
QEMU_AUDIO_DRV=pa
HOSTTYPE=x86_64
FROM_HEADER=
PAGER=less
CSHEDIT=emacs
XDG_CONFIG_DIRS=/etc/xdg
LIBGL_DEBUG=quiet
MINICOM=-c on
MAIL=/var/mail/trainee
PATH=/home/trainee/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
CPU=x86_64
SSH_SENDS_LOCALE=yes
INPUTRC=/home/trainee/.inputrc
PWD=/home/trainee
LANG=fr_FR.UTF-8
PYTHONSTARTUP=/etc/pythonstart
GPG_TTY=/dev/pts/0
AUDIODRIVER=pulseaudio
QT_SYSTEM_DIR=/usr/share/desktop-data
SHLVL=1
HOME=/home/trainee
ALSA_CONFIG_PATH=/etc/alsa-pulse.conf
SDL_AUDIODRIVER=pulse
LESS_ADVANCED_PREPROCESSOR=no
OSTYPE=linux
LS_OPTIONS=-N --color=tty -T 0
XCURSOR_THEME=DMZ
WINDOWMANAGER=env GNOME_SHELL_SESSION_MODE=sle-classic gnome-session --session sle-classic
G_FILENAME_ENCODING=@locale,UTF-8,ISO-8859-15,CP1252
LESS=-M -I -R
MACHTYPE=x86_64-suse-linux
LOGNAME=trainee
XDG_DATA_DIRS=/usr/share
```

```
SSH_CONNECTION=10.0.2.2 46258 10.0.2.15 22
LESSOPEN=lessopen.sh %s
XDG_RUNTIME_DIR=/run/user/1000
NO_AT_BRIDGE=1
LESSCLOSE=lessclose.sh %s %s
G_BROKEN_FILENAMES=1
COLORTERM=1
_=/usr/bin/env
OLDPWD=/home/trainee/training
```

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@SLES12SP1:~> 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@SLES12SP1:~> set -o allexport
trainee@SLES12SP1:~> set -o
allexport       on
braceexpand     on
...
```

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

```
trainee@SLES12SP1:~> set +o allexport
trainee@SLES12SP1:~> 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@SLES12SP1:~> set -o noclobber
trainee@SLES12SP1:~> pwd > file
trainee@SLES12SP1:~> pwd > file
-bash: file: cannot overwrite existing file
trainee@SLES12SP1:~> pwd >| file
trainee@SLES12SP1:~> set +o noclobber
```

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

noglob

```
trainee@SLES12SP1:~> set -o noglob
trainee@SLES12SP1:~> echo *
*
trainee@SLES12SP1:~> set +o noglob
trainee@SLES12SP1:~> echo *
aac abc bca bin codes Desktop Documents Downloads errorlog file file1 Music Pictures Public public_html 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@SLES12SP1:~> set -o nounset
trainee@SLES12SP1:~> echo $FENESTROS
-bash: FENESTROS: unbound variable
trainee@SLES12SP1:~> set +o nounset
trainee@SLES12SP1:~> echo $FENESTROS

trainee@SLES12SP1:~>
```

[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@SLES12SP1:~> script
Script started, file is typescript
trainee@SLES12SP1:~> pwd
/home/trainee
trainee@SLES12SP1:~> ls
aac  bin      Documents  file1  file1     Public       training    vitext
abc  codes    Downloads  file2  Music     public_html  typescript  xyz
bca  Desktop  errorlog   file      Pictures  Templates    Videos
trainee@SLES12SP1:~> exit
exit
Script done, file is typescript
trainee@SLES12SP1:~> cat typescript
Script started on Tue 29 Nov 2016 03:59:24 CET
trainee@SLES12SP1:~> pwd
/home/trainee
trainee@SLES12SP1:~> ls
aac  bin      Documents  file1  file1     Public       training    vitext
abc  codes    Downloads  file2  Music     public_html  typescript  xyz
bca  Desktop  errorlog   file      Pictures  Templates    Videos
trainee@SLES12SP1:~> exit
```

```
exit

Script done on Tue 29 Nov 2016 03:59:31 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@SLES12SP1:~> vi myscript
trainee@SLES12SP1:~> /bin/bash myscript
/home/trainee
aac  bin      Documents  file1  file1      Pictures    Templates   Videos
abc  codes    Downloads  file2  myscript  Public       training    vitext
bca  Desktop  errorlog   file       Music      public_html  typescript  xyz
```

Using a redirection:

```
trainee@SLES12SP1:~> /bin/bash < myscript
/home/trainee
aac  bin      Documents  file1  file1      Pictures    Templates   Videos
abc  codes    Downloads  file2  myscript  Public       training    vitext
```

```
bca  Desktop  errorlog   file       Music      public_html  typescript  xyz
```

In order to be able to call the script by it's name from another directory, sutraich 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/nee/bin**:

```
trainee@SLES12SP1:~> echo $PATH
/home/trainee/bin:/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games
```

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

```
trainee@SLES12SP1:~> mv myscript ~/bin
trainee@SLES12SP1:~> chmod u+x ~/bin/myscript
```

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

```
trainee@SLES12SP1:/tmp> myscript
/tmp
hsperfdata_root
inode
managera141126784165771523 5client
managera3336001029897679475server
managera484793894223296 4844client
managera505035701634772 1452server
systemd-private-04f820fa26c745be8ddba814c6292f21-rtkit-daemon.service-o4lKP5
systemicontmp557867747224513 4133dat
systemicontmp708239220502080 2884dat
```

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

- ./myscript
- . myscript

```
trainee@SLES12SP1:/tmp> cd ~/bin
```

```
trainee@SLES12SP1:~/bin> ./myscript
/home/trainee/bin
myscript
trainee@SLES12SP1:~/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

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@SLES12SP1:~/bin> read var1 var2 var3 var4
fenestros edu is great!
trainee@SLES12SP1:~/bin> echo $var1
fenestros
trainee@SLES12SP1:~/bin> echo $var2
edu
trainee@SLES12SP1:~/bin> echo $var3
is
trainee@SLES12SP1:~/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@SLES12SP1:~/bin> read var1 var2
fenestros edu is great!
trainee@SLES12SP1:~/bin> echo $var1
fenestros
```

```
trainee@SLES12SP1:~/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@SLES12SP1:~/bin> read var
```

↵ Enter

```
trainee@SLES12SP1:~/bin> echo $?
0
trainee@SLES12SP1:~/bin> echo $var

trainee@SLES12SP1:~/bin>
```

But not null:

```
trainee@SLES12SP1:~/bin> read var
```

Ctrl+D

```
trainee@SLES12SP1:~/bin> echo $?
1
trainee@SLES12SP1:~/bin> echo $var

trainee@SLES12SP1:~/bin>
```

The IFS Variable

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

```
trainee@SLES12SP1:~/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@SLES12SP1:~/bin> OLDIFS="$IFS"
trainee@SLES12SP1:~/bin> IFS=":"
trainee@SLES12SP1:~/bin> echo "$IFS" | od -c
0000000   :  \n
0000002
```

Now test the new configuration:

```
trainee@SLES12SP1:~/bin> read var1 var2 var3
fenestros:edu is:great!
trainee@SLES12SP1:~/bin> echo $var1
fenestros
trainee@SLES12SP1:~/bin> echo $var2
edu is
trainee@SLES12SP1:~/bin> echo $var3
great!
```

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

```
trainee@SLES12SP1:~/bin> IFS="$OLDIFS"
```

```
trainee@SLES12SP1:~/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@SLES12SP1:~/bin> cd ../training/
trainee@SLES12SP1:~/training> test -f a100
trainee@SLES12SP1:~/training> echo $?
0
trainee@SLES12SP1:~/training> [ -f a100 ]
trainee@SLES12SP1:~/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@SLES12SP1:~/training> [ -f a101 ]
trainee@SLES12SP1:~/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@SLES12SP1:~/training> [ -d /home/trainee/training ]
trainee@SLES12SP1:~/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

Test	Description
string1	Returns true if string1 is not empty

LAB #2

Test whether two strings are indentical:

```
trainee@SLES12SP1:~/training> string1="root"
trainee@SLES12SP1:~/training> string2="fenestros"
trainee@SLES12SP1:~/training> [ $string1 = $string2 ]
trainee@SLES12SP1:~/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@SLES12SP1:~/training> [ -n $string1 ]
trainee@SLES12SP1:~/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@SLES12SP1:~/training> [ -z $string1 ]
trainee@SLES12SP1:~/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@SLES12SP1:~/training> read value1
35
trainee@SLES12SP1:~/training> read value2
23
trainee@SLES12SP1:~/training> [ $value1 -lt $value2 ]
trainee@SLES12SP1:~/training> echo $?
1
trainee@SLES12SP1:~/training> [ $value2 -lt $value1 ]
trainee@SLES12SP1:~/training> echo $?
0
trainee@SLES12SP1:~/training> [ $value2 -eq $value1 ]
trainee@SLES12SP1:~/training> echo $?
1
```

Expressions

Test	Description
!expression	Returns true if expression is false

Test	Description
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@SLES12SP1:~/training> file=a100
trainee@SLES12SP1:~/training> [ ! -d $file ]
trainee@SLES12SP1:~/training> echo $?
0
```

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

```
trainee@SLES12SP1:~/training> directory=/usr
trainee@SLES12SP1:~/training> [ -d $directory -a -x $directory ]
trainee@SLES12SP1:~/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@SLES12SP1:~/training> [ -w a100 -a \( -d /usr -o -d /tmp \) ]
trainee@SLES12SP1:~/training> echo $?
0
```

Testing the User Environment

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

LAB #5

```
trainee@SLES12SP1:~/training> [ -o allexport ]
trainee@SLES12SP1:~/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@SLES12SP1:~/training> [[ -w a100 && ( -d /usr || -d /tmp ) ]]
trainee@SLES12SP1:~/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@SLES12SP1:~/training> [[ -d /root ]] && echo "The root directory exists"
The root directory exists
trainee@SLES12SP1:~/training> [[ -d /root ]] || echo "The root directory exists"
trainee@SLES12SP1:~/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@SLES12SP1:~/training> x=2
trainee@SLES12SP1:~/training> expr $x + 2
```

```
4
```

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

```
trainee@SLES12SP1:~/training> expr $x+2
2+2
```

Certain operators need to be protected:

```
trainee@SLES12SP1:~/training> expr $x * 2
expr: syntax error
trainee@SLES12SP1:~/training> expr $x \* 2
4
```

Now put the result of a calculation in a variable:

```
trainee@SLES12SP1:~/training> resultat=`expr $x + 10`
trainee@SLES12SP1:~/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

Opérateur	Description
&	Binary AND
|	Binary OR
^	Exclusive binary OR

LAB #9

```
trainee@SLES12SP1:~/training> x=2
trainee@SLES12SP1:~/training> ((x=$x+10))
trainee@SLES12SP1:~/training> echo $x
12
trainee@SLES12SP1:~/training> ((x=$x+20))
trainee@SLES12SP1:~/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 case of SLES, 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@SLES12SP1:~/training> cat ~/.profile
# Sample .profile for SuSE Linux
# rewritten by Christian Steinruecken <cstein@suse.de>
#
# This file is read each time a login shell is started.
```

```
# All other interactive shells will only read .bashrc; this is particularly
# important for language settings, see below.

test -z "$PROFILEREAD" && . /etc/profile || true

# Most applications support several languages for their output.
# To make use of this feature, simply uncomment one of the lines below or
# add your own one (see /usr/share/locale/locale.alias for more codes)
# This overwrites the system default set in /etc/sysconfig/language
# in the variable RC_LANG.
#
#export LANG=de_DE.UTF-8    # uncomment this line for German output
#export LANG=fr_FR.UTF-8    # uncomment this line for French output
#export LANG=es_ES.UTF-8    # uncomment this line for Spanish output


# Some people don't like fortune. If you uncomment the following lines,
# you will have a fortune each time you log in ;-)

#if [ -x /usr/bin/fortune ] ; then
#    echo
#    /usr/bin/fortune
#    echo
#fi
```

~/.bashrc

```
trainee@SLES12SP1:~/training> cat ~/.bashrc
# Sample .bashrc for SuSE Linux
# Copyright (c) SuSE GmbH Nuernberg

# There are 3 different types of shells in bash: the login shell, normal shell
# and interactive shell. Login shells read ~/.profile and interactive shells
```

```
# read ~/.bashrc; in our setup, /etc/profile sources ~/.bashrc - thus all
# settings made here will also take effect in a login shell.
#
# NOTE: It is recommended to make language settings in ~/.profile rather than
# here, since multilingual X sessions would not work properly if LANG is over-
# ridden in every subshell.

# Some applications read the EDITOR variable to determine your favourite text
# editor. So uncomment the line below and enter the editor of your choice :-)
#export EDITOR=/usr/bin/vim
#export EDITOR=/usr/bin/mcedit

# For some news readers it makes sense to specify the NEWSSERVER variable here
#export NEWSSERVER=your.news.server

# If you want to use a Palm device with Linux, uncomment the two lines below.
# For some (older) Palm Pilots, you might need to set a lower baud rate
# e.g. 57600 or 38400; lowest is 9600 (very slow!)
#
#export PILOTPORT=/dev/pilot
#export PILOTRATE=115200

test -s ~/.alias && . ~/.alias || true
```

<html>

</html>