Version : **2023.01**

Updated : 2023/07/27 13:56

LCE405 - Command Line Interface

Contents

        - noglob
        - nounset
    - Basic Shell Scripting
        - Execution
        - The read command
        - The test Command
        - The [[expression]] Command
        - Shell Operators
        - The expr Command
        - The let Command
        - Control Structures
        - Loops
        - Start-up Scripts
        - LAB #1 - Start-up Scripts

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

In RHEL/CentOS 8 **/bin/sh** is a soft link to **/bin/bash** :

```
[trainee@centos8 ~]$ ls -l /bin/sh
lrwxrwxrwx. 1 root root 4 Jul 21  2020 /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:

```
[trainee@centos7 ~]$ type cd
cd is a shell builtin
```

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

```
[trainee@centos8 ~]$ type cd
cd is a shell builtin
```

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@centos8 ~]$ type ls
ls is aliased to `ls --color=auto'
```

> **Important**: Note that the **ls** alias is an alias to the **ls** command itself.

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

```
[trainee@centos8 ~]$ alias dir='ls -l'
[trainee@centos8 ~]$ dir
total 0
-rw-rw-r--. 1 trainee trainee 0 Apr 20 03:46 aac
-rw-rw-r--. 1 trainee trainee 0 Apr 20 03:46 abc
-rw-rw-r--. 1 trainee trainee 0 Apr 20 03:46 bca
-rw-rw-r--. 1 trainee trainee 0 Apr 20 03:46 xyz
```

> **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.

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

```
[trainee@centos8 ~]$ alias
alias dir='ls -l'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias vi='vim'
alias which='(alias; declare -f) | /usr/bin/which --tty-only --read-alias --read-functions --show-tilde --show-
dot'
alias xzegrep='xzegrep --color=auto'
alias xzfgrep='xzfgrep --color=auto'
alias xzgrep='xzgrep --color=auto'
alias zegrep='zegrep --color=auto'
alias zfgrep='zfgrep --color=auto'
alias zgrep='zgrep --color=auto'
```

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.

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

```
[trainee@centos8 ~]$ \dir
aac  abc  bca  xyz
```

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

```
[trainee@centos8 ~]$ unalias dir
[trainee@centos8 ~]$ dir
```

```
aac  abc  bca  xyz
```

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

```
[trainee@centos8 ~]$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:65534:65534:Kernel Overflow User:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
systemd-coredump:x:999:997:systemd Core Dumper:/:/sbin/nologin
systemd-resolve:x:193:193:systemd Resolver:/:/sbin/nologin
tss:x:59:59:Account used by the trousers package to sandbox the tcsd daemon:/dev/null:/sbin/nologin
polkitd:x:998:996:User for polkitd:/:/sbin/nologin
unbound:x:997:994:Unbound DNS resolver:/etc/unbound:/sbin/nologin
libstoragemgmt:x:996:993:daemon account for libstoragemgmt:/var/run/lsm:/sbin/nologin
cockpit-ws:x:995:991:User for cockpit-ws:/nonexisting:/sbin/nologin
sssd:x:994:990:User for sssd:/:/sbin/nologin
setroubleshoot:x:993:989::/var/lib/setroubleshoot:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
chrony:x:992:988::/var/lib/chrony:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
trainee:x:1000:1000:trainee:/home/trainee:/bin/bash
cockpit-wsinstance:x:991:987:User for cockpit-ws instances:/nonexisting:/sbin/nologin
rngd:x:990:986:Random Number Generator Daemon:/var/lib/rngd:/sbin/nologin
```

```
gluster:x:989:985:GlusterFS daemons:/run/gluster:/sbin/nologin
qemu:x:107:107:qemu user:/:/sbin/nologin
rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
saslauth:x:988:76:Saslauthd user:/run/saslauthd:/sbin/nologin
radvd:x:75:75:radvd user:/:/sbin/nologin
dnsmasq:x:983:983:Dnsmasq DHCP and DNS server:/var/lib/dnsmasq:/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@centos8 ~]$ cat /etc/shells
/bin/sh
/bin/bash
/usr/bin/sh
/usr/bin/bash
```

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

```
[trainee@centos8 ~]$ echo $SHELL
/bin/bash
```

> **Important** : Note that when using RHEL/CentOS 7 the output shows that trainee's shell is **/bin/bash** and not **/usr/bin/bash**. This is because /bin is a soft link to /usr/bin.

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

```
[trainee@centos8 ~]$ chsh
Changing shell for trainee.
New shell [/bin/bash]
/bin/sh
Password: trainee
```

```
Shell changed.
```

Important: Note that the password will not be printed to standard output.

Now check your current shell:

```
[trainee@centos8 ~]$ 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@centos8 ~]$ cat /etc/passwd | grep trainee
trainee:x:1000:1000:trainee:/home/trainee:/bin/sh
```

Important : The /bin/sh shell will be your active shell the next time you login.

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

```
[trainee@centos8 ~]$ chsh
Changing shell for trainee.
New shell [/bin/sh]: /bin/bash
Password: trainee
Shell changed.
```

Important: Note that the password will not be printed to standard output.

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@centos8 ~]$ history | more
    1  su -
    2  exit
    3  su -
    4  nmcli c show
    5  stty -a
    6  date
    7  who
    8  df
    9  df -h
   10  free free -h
   11  free
   12  free -h
   13  whoami
   14  su -
   15  pwd
   16  cd /tmp
   17  pwd
   18  ls
   19  su -
   20  touch test
```

```
   21  ls
   22  echo fenestros
   23  cp test ~
--More--
```

Important: The history is specific to each user.

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@centos8 ~]$ ls
aac  abc  bca  xyz
[trainee@centos8 ~]$ !!
ls
aac  abc  bca  xyz
```

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

```
[trainee@centos8 ~]$ history
    1  su -
...
   80  history | more
   81  ls
   82  history
[trainee@centos8 ~]$ !81
ls
aac  abc  bca  xyz
```

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

```
[trainee@centos8 ~]$ cat /etc/profile | grep HISTSIZE
HISTSIZE=1000
export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE HISTCONTROL
```

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@centos8 ~]$ nl .bash_history | tail
    54  ls
    55  ls | sort
    56  ls | sort -r
    57  more /etc/services
```

```
    58  less /etc/services
    59  find acc
    60  find aac
    61  su -
    62  sleep 10
    63  su -
```

> **Important** : Note the use of the **nl** command to number the lines in the output of the contents of **.bash_history** file.

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@centos8 ~]$ ls .bash
.bash_history  .bash_logout   .bash_profile  .bashrc
```

> **Important** : Notez qu'en appuyant sur la touche Tab trois fois le shell propose 4 possibilités de complétion de nom de fichier. En effet, sans plus d'information, le shell ne sait pas quel fichier est concerné.

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@centos8 ~]$ mo
modinfo                 more                    mount.nfs4
modprobe                mount                   mountpoint
modulemd-validator      mount.fuse              mountstats
modulemd-validator-v1   mount.nfs
```

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@centos8 ~]$ mkdir training
```

```
[trainee@centos8 ~]$ cd training
[trainee@centos8 training]$ touch f1 f2 f3 f4 f5
[trainee@centos8 training]$ ls
f1  f2  f3  f4  f5
```

The * Metacharacter

Now use the Metacharacter *:

```
[trainee@centos8 training]$ echo f*
f1 f2 f3 f4 f5
```

Important: Note that the * is used as a wild card which replaces 0 or more characters.

The ? Metacharacter

Create two more files:

```
[trainee@centos8 training]$ touch f52 f62
```

Now use the Metacharacter **?**:

```
[trainee@centos8 training]$ echo f?2
f52 f62
```

Important: Note that the **?** is used as a wild card which replaces a single character.

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@centos8 training]$ touch a100
```

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

```
[trainee@centos8 training]$ echo [a-f]*
a100 f1 f2 f3 f4 f5 f52 f62
[trainee@centos8 training]$ echo [af]*
a100 f1 f2 f3 f4 f5 f52 f62
```

Important: Note that all the files starting with either **a**, **b**, **c**, **d**, **e** or **f** are displayed.

```
[trainee@centos8 training]$ echo [!a]*
f1 f2 f3 f4 f5 f52 f62
```

Important: Note that all the files in the directory are displayed except the file starting with **a** .

```
[trainee@centos8 training]$ echo [a-b]*
a100
```

Important: Note that only the file starting with **a** is displayed since no file starting with **b** is present.

```
[trainee@centos8 training]$ echo [a-f]
[a-f]
```

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.

The extglob Option

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

```
[trainee@centos8 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@centos8 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        of
```

?(expression)

Create the following files:

```
[trainee@centos8 training]$ touch f f.txt f123.txt f123123.txt f123123123.txt
```

Execute the following command:

```
[trainee@centos8 training]$ ls f?(123).txt
f123.txt  f.txt
```

> **Important**: Note that the command displays file names that match 0 or 1 occurrences of the string **123**.

***(expression)**

Execute the following command:

```
[trainee@centos8 training]$ ls f*(123).txt
f123123123.txt  f123123.txt  f123.txt  f.txt
```

Important: Note that the command displays file names that match 0 to x occurrences of the string **123**.

+(expression)

Execute the following command:

```
[trainee@centos8 training]$ ls f+(123).txt
f123123123.txt  f123123.txt  f123.txt
```

Important: Note that the command displays file names that match 1 to x occurrences of the string **123**..

@(expression)

Execute the following command:

```
[trainee@centos8 training]$ ls f@(123).txt
f123.txt
```

Important: Note that the command displays file names that match 1 occurrence of the string **123**.

!(expression)

Execute the following command:

```
[trainee@centos8 training]$ ls f!(123).txt
f123123123.txt  f123123.txt  f.txt
```

> **Important**: Note that the command displays file names that match 0 or x occurrences of the string **123**, where x>1.

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@centos8 training]$ echo * is a metacharacter
a100 f f1 f123123123.txt f123123.txt f123.txt f2 f3 f4 f5 f52 f62 f.txt est un caractère spécial

[trainee@centos8 training]$ echo \* is a metacharacter
* is a metacharacter

[trainee@centos8 training]$ echo "* is a metacharacter"
* is a metacharacter
```

```
[trainee@centos8 training]$ echo '* is a metacharacter'
* is a metacharacter
```

Exit Status

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

For example:

```
[trainee@centos8 training]$ cd ..
[trainee@centos8 ~]$ mkdir codes
[trainee@centos8 ~]$ echo $?
0
[trainee@centos8 ~]$ touch codes/exit.txt
[trainee@centos8 ~]$ rmdir codes
rmdir: failed to remove ‘codes’: Directory not empty
[trainee@centos8 ~]$ 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@centos8 ~]$ pwd
/home/trainee
[trainee@centos8 ~]$ cd training
[trainee@centos8 training]$ free > file
[trainee@centos8 training]$ cat file
              total        used        free      shared  buff/cache   available
Mem:         500780      192692       38916        4824      269172      260472
Swap:       2096124           0     2096124
```

Important: If the file does not exist, it is automatically created.

Repeating a single redirection will replace the file:

```
[trainee@centos8 training]$ date > file
[trainee@centos8 training]$ cat file
Mon 28 Nov 15:48:09 CET 2016
```

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

```
[trainee@centos8 training]$ free >> file
[trainee@centos8 training]$ cat file
Mon 28 Nov 15:48:09 CET 2016
              total        used        free      shared  buff/cache   available
Mem:         500780      192792       38516        4824      269472      260376
Swap:       2096124           0     2096124
```

Important : Note that standard output can only be redirected to a single destination.

File descriptors are numbered for ease of use :

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

For example:

```
[trainee@centos8 training]$ cd ..
[trainee@centos8 ~]$ rmdir training/ 2>errorlog
[trainee@centos8 ~]$ cat errorlog
rmdir: failed to remove ‘training/’: Directory not empty
```

Important: As you can see the error generated is redirected to the **errorlog** file.

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

```
[trainee@centos8 ~]$ 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@centos8 ~]$ 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@centos8 ~]$ ls | wc -w
7
```

Important - Several pipes can be used within the same command.

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

```
[trainee@centos8 ~]$ date | tee file1
Tue 20 Apr 10:39:47 EDT 2021
[trainee@centos8 ~]$ cat file1
Tue 20 Apr 10:39:47 EDT 2021
```

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

```
[trainee@centos8 ~]$ date | tee file1 > file2
[trainee@centos8 ~]$ cat file1
Tue 20 Apr 10:40:36 EDT 2021
[trainee@centos8 ~]$ cat file2
Tue 20 Apr 10:40:36 EDT 2021
```

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.

Command Substitution

Command substitution permits in-line execution of a command:

```
[trainee@centos8 ~]$ echo date
date
[trainee@centos8 ~]$ echo $(date)
Tue 20 Apr 10:41:33 EDT 2021
[trainee@centos8 ~]$ echo `date`
Tue 20 Apr 10:41:45 EDT 2021
```

Conditional Command Execution

Commands can be grouped using brackets:

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

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.

Variable	Description
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@centos8 ~]$ echo $LC_ALL

[trainee@centos8 ~]$ echo $LC_CTYPE

[trainee@centos8 ~]$ echo $LANG
en_GB.UTF-8
[trainee@centos8 ~]$ 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=
```

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

Variable	Description
$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 Command

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@centos8 ~]$ env
LS_COLORS=rs=0:di=38;5;33:ln=38;5;51:mh=00:pi=40;38;5;11:so=38;5;13:do=38;5;5:bd=48;5;232;38;5;11:cd=48;5;232;38;
5;3:or=48;5;232;38;5;9:mi=01;05;37;41:su=48;5;196;38;5;15:sg=48;5;11;38;5;16:ca=48;5;196;38;5;226:tw=48;5;10;38;5
;16:ow=48;5;10;38;5;21:st=48;5;21;38;5;15:ex=38;5;40:*.tar=38;5;9:*.tgz=38;5;9:*.arc=38;5;9:*.arj=38;5;9:*.taz=38
;5;9:*.lha=38;5;9:*.lz4=38;5;9:*.lzh=38;5;9:*.lzma=38;5;9:*.tlz=38;5;9:*.txz=38;5;9:*.tzo=38;5;9:*.t7z=38;5;9:*.z
ip=38;5;9:*.z=38;5;9:*.dz=38;5;9:*.gz=38;5;9:*.lrz=38;5;9:*.lz=38;5;9:*.lzo=38;5;9:*.xz=38;5;9:*.zst=38;5;9:*.tzs
t=38;5;9:*.bz2=38;5;9:*.bz=38;5;9:*.tbz=38;5;9:*.tbz2=38;5;9:*.tz=38;5;9:*.deb=38;5;9:*.rpm=38;5;9:*.jar=38;5;9:*
.war=38;5;9:*.ear=38;5;9:*.sar=38;5;9:*.rar=38;5;9:*.alz=38;5;9:*.ace=38;5;9:*.zoo=38;5;9:*.cpio=38;5;9:*.7z=38;5
;9:*.rz=38;5;9:*.cab=38;5;9:*.wim=38;5;9:*.swm=38;5;9:*.dwm=38;5;9:*.esd=38;5;9:*.jpg=38;5;13:*.jpeg=38;5;13:*.mj
pg=38;5;13:*.mjpeg=38;5;13:*.gif=38;5;13:*.bmp=38;5;13:*.pbm=38;5;13:*.pgm=38;5;13:*.ppm=38;5;13:*.tga=38;5;13:*.
xbm=38;5;13:*.xpm=38;5;13:*.tif=38;5;13:*.tiff=38;5;13:*.png=38;5;13:*.svg=38;5;13:*.svgz=38;5;13:*.mng=38;5;13:*
.pcx=38;5;13:*.mov=38;5;13:*.mpg=38;5;13:*.mpeg=38;5;13:*.m2v=38;5;13:*.mkv=38;5;13:*.webm=38;5;13:*.ogm=38;5;13:
*.mp4=38;5;13:*.m4v=38;5;13:*.mp4v=38;5;13:*.vob=38;5;13:*.qt=38;5;13:*.nuv=38;5;13:*.wmv=38;5;13:*.asf=38;5;13:*
.rm=38;5;13:*.rmvb=38;5;13:*.flc=38;5;13:*.avi=38;5;13:*.fli=38;5;13:*.flv=38;5;13:*.gl=38;5;13:*.dl=38;5;13:*.xc
f=38;5;13:*.xwd=38;5;13:*.yuv=38;5;13:*.cgm=38;5;13:*.emf=38;5;13:*.ogv=38;5;13:*.ogx=38;5;13:*.aac=38;5;45:*.au=
38;5;45:*.flac=38;5;45:*.m4a=38;5;45:*.mid=38;5;45:*.midi=38;5;45:*.mka=38;5;45:*.mp3=38;5;45:*.mpc=38;5;45:*.ogg
=38;5;45:*.ra=38;5;45:*.wav=38;5;45:*.oga=38;5;45:*.opus=38;5;45:*.spx=38;5;45:*.xspf=38;5;45:
SSH_CONNECTION=10.0.2.2 42834 10.0.2.15 22
LANG=en_GB.UTF-8
HISTCONTROL=ignoredups
GUESTFISH_RESTORE=\e[0m
```

```
HOSTNAME=centos8.ittraining.loc
GUESTFISH_INIT=\e[1;34m
XDG_SESSION_ID=9
USER=trainee
GUESTFISH_PS1=\[\e[1;32m\]><fs>\[\e[0;31m\]
SELINUX_ROLE_REQUESTED=
PWD=/home/trainee
HOME=/home/trainee
SSH_CLIENT=10.0.2.2 42834 22
SELINUX_LEVEL_REQUESTED=
SSH_TTY=/dev/pts/0
MAIL=/var/spool/mail/trainee
TERM=xterm-256color
SHELL=/bin/bash
SELINUX_USE_CURRENT_RANGE=
SHLVL=1
LOGNAME=trainee
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
XDG_RUNTIME_DIR=/run/user/1000
PATH=/home/trainee/.local/bin:/home/trainee/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
GUESTFISH_OUTPUT=\e[0m
HISTSIZE=1000
LESSOPEN=||/usr/bin/lesspipe.sh %s
_=/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@centos8 ~]$ 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@centos8 ~]$ set -o allexport
[trainee@centos8 ~]$ set -o
allexport       on
braceexpand     on
...
```

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

```
[trainee@centos8 ~]$ set +o allexport
[trainee@centos8 ~]$ 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@centos8 ~]$ set -o noclobber
[trainee@centos8 ~]$ pwd > file
-bash: file: cannot overwrite existing file
[trainee@centos8 ~]$ pwd > file
```

```
-bash: file: cannot overwrite existing file
[trainee@centos8 ~]$ pwd >| file
[trainee@centos8 ~]$ set +o noclobber
```

Important : Note that the **noclobber** option can be overidden by using a pipe.

noglob

```
[trainee@centos8 ~]$ set -o noglob
[trainee@centos8 ~]$ echo *
*
[trainee@centos8 ~]$ set +o noglob
[trainee@centos8 ~]$ echo *
aac abc bca codes Desktop Documents Downloads errorlog file file1 Music Pictures Public Templates training Videos
vitext xyz
```

Important : Note that metacharacters are turned off when the **noglob** option is set.

nounset

```
[trainee@centos8 ~]$ set -o nounset
[trainee@centos8 ~]$ echo $FENESTROS
-bash: FENESTROS: unbound variable
[trainee@centos8 ~]$ set +o nounset
[trainee@centos8 ~]$ echo $FENESTROS
```

```
[trainee@centos8 ~]$
```

Important : Note that the inexistant variable **$FENESTROS** is identified as such when the **nounset** option is set.

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**

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.

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@centos8 ~]$ script
Script started, file is typescript
[trainee@centos8 ~]$ pwd
/home/trainee
[trainee@centos8 ~]$ ls
aac  abc  bca  codes  errorlog  file  file1  file2  training  typescript  xyz
[trainee@centos8 ~]$ exit
exit
Script done, file is typescript

[trainee@centos8 ~]$ cat typescript
Script started on 2021-04-20 10:59:58-04:00
[trainee@centos8 ~]$ pwd
/home/trainee
[trainee@centos8 ~]$ ls
aac  abc  bca  codes  errorlog  file  file1  file2  training  typescript  xyz
[trainee@centos8 ~]$ exit
exit

Script done on 2021-04-20 11:00:09-04:00
```

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

```
[trainee@centos8 ~]$ vi myscript
[trainee@centos8 ~]$ cat myscript
pwd
ls
```

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

As an argument de /bin/bash:

```
[trainee@centos8 ~]$ /bin/bash myscript
/home/trainee
aac  bca    errorlog  file1  myscript  typescript
abc  codes  file      file2  training  xyz
```

Using a redirection:

```
[trainee@centos8 ~]$ /bin/bash < myscript
/home/trainee
aac  bca    errorlog  file1  myscript  typescript
abc  codes  file      file2  training  xyz
```

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@centos8 ~]$ echo $PATH
/home/trainee/.local/bin:/home/trainee/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
```

In the case of RHEL/CentOS, even though PATH contains $HOME/bin, the directory is not present:

```
[trainee@centos8 ~]$ ls
aac  bca    errorlog  file1  myscript  typescript
```

```
abc  codes  file      file2  training  xyz
```

So you need to create the directory:

```
[trainee@centos8 ~]$ mkdir bin
```

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

```
[trainee@centos8 ~]$ mv myscript ~/bin
[trainee@centos8 ~]$ chmod u+x ~/bin/myscript
```

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

```
[trainee@centos8 ~]$ cd /tmp
[trainee@centos8 tmp]$ myscript
/tmp
expand
expand1
filepartaa
filepartab
filepartac
filepartad
filepartae
greptest
greptest1
greptest.patch
newfile
sales.awk
sales.txt
scriptawk
sedtest
sedtest1
systemd-private-d9ff2376a8a44f0392f860d80c839be4-chronyd.service-6im4Ii
```

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

```
[trainee@centos8 tmp]$ cd ~/bin
[trainee@centos8 bin]$ ./myscript
/home/trainee/bin
myscript
[trainee@centos8 bin]$ . myscript
/home/trainee/bin
myscript
```

To do: Note the difference in the output of these two commands and explain that difference.

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@centos8 bin]$ read var1 var2 var3 var4
fenestros edu is great!
[trainee@centos8 bin]$ echo $var1
fenestros
[trainee@centos8 bin]$ echo $var2
edu
[trainee@centos8 bin]$ echo $var3
is
[trainee@centos8 bin]$ echo $var4
great!
```

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.

```
[trainee@centos8 bin]$ read var1 var2
fenestros edu is great!
[trainee@centos8 bin]$ echo $var1
fenestros
[trainee@centos8 bin]$ echo $var2
edu is great!
```

Important: Note that in this case, $var2 contains three fields.

Exit Codes

The contents of a variable can also be empty:

```
[trainee@centos8 bin]$ read var
```

↵ Entrée

```
[trainee@centos8 bin]$ echo $?
0
[trainee@centos8 bin]$ echo $var

[trainee@centos8 bin]$
```

But not null:

```
[trainee@centos8 bin]$ read var
```

Ctrl+D

```
[trainee@centos8 bin]$ echo $?
1
[trainee@centos8 bin]$ echo $var

[trainee@centos8 bin]$
```

The IFS Variable

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

```
[trainee@centos8 bin]$ echo "$IFS" | od -c
0000000      \t  \n  \n
0000004
```

> **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.

It is possible to change the contents of this variable:

```
[trainee@centos8 bin]$ OLDIFS="$IFS"
[trainee@centos8 bin]$ IFS=":"
[trainee@centos8 bin]$ echo "$IFS" | od -c
0000000   :  \n
```

```
0000002
```

Now test the new configuration:

```
[trainee@centos8 bin]$ read var1 var2 var3
fenestros:edu is:great!
[trainee@centos8 bin]$ echo $var1
fenestros
[trainee@centos8 bin]$ echo $var2
edu is
[trainee@centos8 bin]$ echo $var3
great!
```

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

```
[trainee@centos8 bin]$ IFS="$OLDIFS"
[trainee@centos8 bin]$ 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

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

```
[trainee@centos8 bin]$ cd ../training/
[trainee@centos8 training]$ test -f a100
[trainee@centos8 training]$ echo $?
0
[trainee@centos8 training]$ [ -f a100 ]
[trainee@centos8 training]$ echo $?
0
```

Important: The value contained in $? is 0. This indicates **true**.

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

```
[trainee@centos8 training]$ [ -f a101 ]
[trainee@centos8 training]$ echo $?
```

```
1
```

Important: The value contained in $? is 1. This indicates **false**. This is obvious since a101 does not exist.

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

```
[trainee@centos8 training]$ [ -d /home/trainee/training ]
[trainee@centos8 training]$ echo $?
0
```

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

Test whether two strings are indentical:

```
[trainee@centos8 training]$ string1="root"
[trainee@centos8 training]$ string2="fenestros"
[trainee@centos8 training]$ [ $string1 = $string2 ]
[trainee@centos8 training]$ echo $?
1
```

Important: The value contained in $? is 1. This indicates **false**.

Test if string1 is not zero in length:

```
[trainee@centos8 training]$ [ -n $string1 ]
[trainee@centos8 training]$ echo $?
0
```

Important: The value contained in $? is 1. This indicates **false**.

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

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

```
[trainee@centos8 training]$ read value1
35
[trainee@centos8 training]$ read value2
23
[trainee@centos8 training]$ [ $value1 -lt $value2 ]
[trainee@centos8 training]$ echo $?
1
[trainee@centos8 training]$ [ $value2 -lt $value1 ]
[trainee@centos8 training]$ echo $?
0
```

```
[trainee@centos8 training]$ [ $value2 -eq $value1 ]
[trainee@centos8 training]$ 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

Test if $file is not a directory:

```
[trainee@centos8 training]$ file=a100
[trainee@centos8 training]$ [ ! -d $file ]
[trainee@centos8 training]$ echo $?
0
```

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

```
[trainee@centos8 training]$ directory=/usr
[trainee@centos8 training]$ [ -d $directory -a -x $directory ]
[trainee@centos8 training]$ 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@centos8 training]$ [ -w a100 -a \( -d /usr -o -d /tmp \) ]
[trainee@centos8 training]$ echo $?
0
```

Testing the User Environment

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

```
[trainee@centos7 training]$ [ -o allexport ]
[trainee@centos7 training]$ echo $?
1
```

The [[expression]] Command

The **[[**SpaceBar**expression**SpaceBar**]]** 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

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@centos8 training]$ [[ -w a100 && ( -d /usr || -d /tmp ) ]]
[trainee@centos8 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

```
[trainee@centos8 training]$ [[ -d /root ]] && echo "The root directory exists"
The root directory exists
[trainee@centos8 training]$ [[ -d /root ]] || echo "The root directory exists"
[trainee@centos8 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

Operator	Description
-	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

Add two to the value of $x:

```
[trainee@centos8 training]$ x=2
[trainee@centos8 training]$ expr $x + 2
4
```

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

```
[trainee@centos8 training]$ expr $x+2
```

```
2+2
```

Certain operators need to be protected:

```
[trainee@centos8 training]$ expr $x * 2
expr: syntax error
[trainee@centos8 training]$ expr $x \* 2
4
```

Now put the result of a calculation in a variable:

```
[trainee@centos8 training]$ resultat=`expr $x + 10`
[trainee@centos8 training]$ 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

Operator	Description
*	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

For example:

```
[trainee@centos8 training]$ x=2
[trainee@centos8 training]$ ((x=$x+10))
[trainee@centos8 training]$ echo $x
12
[trainee@centos8 training]$ ((x=$x+20))
[trainee@centos8 training]$ 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
```

As an example, create the following script called **user_check**:

```
[trainee@centos8 training]$ vi user_check
[trainee@centos8 training]$ cat user_check
#!/bin/bash
if [ $# -ne 1 ] ; then
  echo "Incorrect number of arguments"
  echo "Usage : $0 user name"
  exit 1
fi
if grep "^$1:" /etc/passwd > /dev/null
then
  echo "User $1 has an account on this system"
else
  echo "User $1 does not have an account on this system"
fi
exit 0
```

Test this script:

```
[trainee@centos8 training]$ chmod 770 user_check
[trainee@centos8 training]$ ./user_check
Incorrect number of arguments
Usage : ./user_check user name
[trainee@centos8 training]$ ./user_check root
User root has an account on this system
[trainee@centos8 training]$ ./user_check mickey mouse
Incorrect number of arguments
Usage : ./user_check user name
[trainee@centos8 training]$ ./user_check "mickey mouse"
User mickey mouse does not have an account on this system
```

case

The syntax is as follows:

```
case $variable in
model1) function
   ...
   ;;
model2) function
   ...
   ;;
model3 | model4 | model5 ) function
   ...
   ;;
esac
```

For example:

```
  case "$1" in
      start)
```

```
        start
        ;;
    stop)
        stop
        ;;
    restart|reload)
        stop
        start
        ;;
    status)
        status
        ;;
    *)
        echo $"Usage: $0 {start|stop|restart|status}"
        exit 1
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 "Account fenestros$U created"
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 RHEL/CentOS, Bash executes **~/.bash_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 #1 - Start-up Scripts

To do : Using the knowledge you have acquired in this unit, explain each of the following scripts.

~/.bash_profile

```
[trainee@centos8 training]$ cat ~/.bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs
```

~/.bashrc

```
[trainee@centos8 training]$ cat ~/.bashrc
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
    . /etc/bashrc
fi

# User specific environment
```

```
PATH="$HOME/.local/bin:$HOME/bin:$PATH"
export PATH

# Uncomment the following line if you don't like systemctl's auto-paging feature:
# export SYSTEMD_PAGER=

# User specific aliases and functions
```
