Version : **2024.01**

Last update : 2024/11/26 10:56

RH12405 - The Command Line Interface

Contents

The Shell

A shell is a **command line interpreter** (C.L.I). It is used as an interface to give instructions or **commands** to the operating system.

The word shell is generic. There are many shells in the Unix world, for example :

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

Under RHEL 9 the **/bin/sh** shell is a symbolic link to **/bin/bash** :

```
[trainee@redhat9 ~]$ ls -l /bin/sh
lrwxrwxrwx. 1 root root 4 Feb 15 2024 /bin/sh -> bash
```

LAB #1 - The /bin/bash Shell

This module is about using the **bash** shell under Linux. The **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 begins with a keyword. This keyword is interpreted by the shell according to the type of command and in the following order:

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

1.1 - Built-in and External Shell Commands

Built-in shell commands are commands such as **cd**. To check the type of command, use the **type** command:

```
[trainee@redhat9 ~]$ type cd
cd is a shell builtin
```

Commands external to the shell are executable binaries or scripts, generally located in /bin, /sbin, /usr/bin or /usr/sbin :

```
[trainee@redhat9 ~]$ type ifconfig
ifconfig is /usr/sbin/ifconfig
```

1.2 - Aliases

Aliases are names used to designate a command or a sequence of commands and are specific only to the shell that created them and to the user's environment :

```
[trainee@redhat9 ~]$ type ls
ls is aliased to `ls --color=auto'
```

> **Important**: Note that in this case the **ls** alias is indeed an alias that uses the ls **command** itself.

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

```
[trainee@redhat9 ~]$ alias dir=‘ls -l’
[trainee@redhat9 ~]$ dir
total 4
-rw-r--r--. 1 trainee trainee 0 Sep 25 15:11 aac
-rw-r--r--. 1 trainee trainee 0 Sep 25 15:11 abc
-rw-r--r--. 1 trainee trainee 0 Sep 25 15:11 bca
drwxr-xr-x. 2 trainee trainee 6 Oct 19 2023 Desktop
drwxr-xr-x. 2 trainee trainee 6 Oct 19 2023 Documents
drwxr-xr-x. 2 trainee trainee 6 Oct 19 2023 Downloads
drwxr-xr-x. 2 trainee trainee 6 Oct 19 2023 Music
drwxr-xr-x. 2 trainee trainee 6 Oct 19 2023 Pictures
drwxr-xr-x. 2 trainee trainee 6 Oct 19 2023 Public
drwxr-xr-x. 2 trainee trainee 6 Oct 19 2023 Templates
drwxr-xr-x. 2 trainee trainee 6 Oct 19 2023 Videos
```

```
-rw-r--r--. 1 trainee trainee 442 Sep 25 14:24 vitext
-rw-r--r--. 1 trainee trainee 0 Sep 25 15:11 xyz
```

Important: Note that the **dir** command actually exists. Creating an alias that is called **dir** implies that the alias will be executed instead of the **dir** command.

The list of defined aliases can be viewed using the **alias** command:

```
[trainee@redhat9 ~]$ 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 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: Note that this list contains, indiscriminately, aliases defined in system startup files as well as the **dir** alias created by **trainee** that is only available to **trainee** in the current terminal.

To force execution of a command and not the alias you must precede the command with the **** character:

```
[trainee@redhat9 ~]$ \dir
aac abc bca Desktop Documents Downloads Music Pictures Public Templates Videos vitext xyz
```

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

```
[trainee@redhat9 ~]$ unalias dir
[trainee@redhat9 ~]$ dir
aac abc bca Desktop Documents Downloads Music Pictures Public Templates Videos vitext xyz
```

1.3 - Defining a user's shell

The user shell is defined by **root** in the last field of the **/etc/passwd** file:

```
[trainee@redhat9 ~]$ 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
systemd-coredump:x:999:997:systemd Core Dumper:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:998:996:User for polkitd:/:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
tss:x:59:59:Account used for TPM access:/dev/null:/sbin/nologin
```

```
colord:x:997:993:User for colord:/var/lib/colord:/sbin/nologin
clevis:x:996:992:Clevis Decryption Framework unprivileged user:/var/cache/clevis:/usr/sbin/nologin
rtkit:x:172:172:RealtimeKit:/proc:/sbin/nologin
sssd:x:995:991:User for sssd:/:/sbin/nologin
geoclue:x:994:990:User for geoclue:/var/lib/geoclue:/sbin/nologin
libstoragemgmt:x:988:988:daemon account for libstoragemgmt:/:/usr/sbin/nologin
systemd-oom:x:987:987:systemd Userspace OOM Killer:/:/usr/sbin/nologin
setroubleshoot:x:986:986:SELinux troubleshoot server:/var/lib/setroubleshoot:/sbin/nologin
pipewire:x:985:984:PipeWire System Daemon:/var/run/pipewire:/sbin/nologin
flatpak:x:984:983:User for flatpak system helper:/:/sbin/nologin
gdm:x:42:42::/var/lib/gdm:/sbin/nologin
cockpit-ws:x:983:982:User for cockpit web service:/nonexisting:/sbin/nologin
cockpit-wsinstance:x:982:981:User for cockpit-ws instances:/nonexisting:/sbin/nologin
gnome-initial-setup:x:981:980::/run/gnome-initial-setup/:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/usr/share/empty.sshd:/sbin/nologin
chrony:x:980:979:chrony system user:/var/lib/chrony:/sbin/nologin
dnsmasq:x:979:978:Dnsmasq DHCP and DNS server:/var/lib/dnsmasq:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
trainee:x:1000:1000:trainee:/home/trainee:/bin/bash
```

However the user can change their shell using the **chsh** command. The shells available to system users are listed in the **/etc/shells** file. Enter the **cat /etc/shells** command:

```
[trainee@redhat9 ~]$ cat /etc/shells
/bin/sh
/bin/bash
/usr/bin/sh
/usr/bin/bash
```

Then use the **echo** command to display the current **trainee** shell:

```
[trainee@redhat9 ~]$ echo $SHELL
/bin/bash
```

Important: Note that under RHEL 9, the system informs us that the current shell of the **trainee** user is **/bin/bash** and not **/usr/bin/bash**. This is because the /bin directory is a symbolic link pointing to the /usr/bin directory.

Then change the **trainee** shell using the **chsh** command, specifying the value of **/bin/sh** for the new shell:

```
[trainee@redhat9 ~]$ chsh
Changing shell for trainee.
New shell [/bin/bash]: /bin/sh
Password: trainee
Shell changed.
```

Important: Note that the password entered will **not** be visible.

Next, check the active shell for **trainee** :

```
[trainee@redhat9 ~]$ echo $SHELL
/bin/bash
```

Lastly check the shell stipulated in the **/etc/passwd** file for **trainee**:

```
[trainee@redhat9 ~]$ cat /etc/passwd | grep trainee
trainee:x:1000:1000:trainee:/home/trainee:/bin/sh
```

Important: You will notice that the active shell is always **/bin/bash** whereas the shell stipulated in the /etc/passwd file is the **/bin/sh**. The **/bin/sh** shell will only become **trainee**'s active shell the next time it connects to the system.

Change your shell to **/bin/bash** again using the chsh command:

```
[trainee@redhat9 ~]$ chsh
Changing shell for trainee.
New shell [/bin/sh]: /bin/bash
Password: trainee
Shell changed.
```

Important: Note that the password entered will **not** be visible.

1.4 - The Prompt

A user's prompt depends on their status:

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

1.5 - Recalling commands

The **/bin/bash** shell can be used to recall the last commands entered. To see the list of memorised commands, use the history command:

```
[trainee@redhat9 ~]$ history | more
    1 su -
    2 exit
    3 su -
    4 clear
    5 cd /
    6 ls -l
    7 cd afs
```

```
    8 ls
    9 cd /
   10 su -
   11 cd ~
   12 vi vitext
   13 view vitext
   14 vi vitext
   15 vi .exrc
   16 vi vitext
   17 su -
   18 stty -a
   19 date
   20 who
   21 df
   22 df -h
   23 free
   24 free -h
   25 whoami
   26 su -
   27 pwd
   28 cd /tmp
   29 pwd
   30 ls
   31 su -
   32 touch test
   33 ls
   34 echo fenestros
   35 cp test ~
   36 ls -l ~
   37 file ~/test
--More--
[q]
```

Important: The history is specific to each user.

The command history is in **emacs** mode by default. As a result, the last command is recalled using the **[Up Arrow]** key or the **[CTRL]-[P]** keys and the next command is recalled using the **[Down Arrow]** key or the **[CTRL]-[N]** keys :

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

To move through the history line :

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

To search the history, use the keys :

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 in the history using the **!!** characters:

```
[trainee@redhat9 ~]$ ls
aac abc bca Desktop Documents Downloads Music Pictures Public Templates Videos vitext xyz
```

```
[trainee@redhat9 ~]$ !!
ls
aac abc bca Desktop Documents Downloads Music Pictures Public Templates Videos vitext xyz
```

You can recall a specific command from the history by using the **!** character followed by the number of the command to be recalled:

```
[trainee@redhat9 ~]$ history
    1 su -
    2 exit
    3 su -
    4 clear
    5 cd /
    6 ls -l
    7 cd afs
    8 ls
    9 cd /
   10 su -
...
   85 echo $SHELL
   86 cat /etc/passwd | grep trainee
   87 chsh
   88 history | more
   89 clear
   90 ls
   91 history

[trainee@redhat9 ~]$ !90
ls
aac abc bca Desktop Documents Downloads Music Pictures Public Templates Videos vitext xyz
```

The command callback function is set up for all users in the **/etc/profile** file. In this file, variables concerning the order callback can be defined. The most important is **HISTSIZE** :

```
[trainee@redhat9 ~]$ cat /etc/profile | grep HISTSIZE
```

```
HISTSIZE=1000
export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE HISTCONTROL
```

You will notice that in the previous case, the value of **HISTSIZE** is **1000**. This implies that the last thousand commands are memorised.

The memorised commands are stored in the **~/.bash_history** file. The commands for the current session are not saved in this file until the session is closed:

```
[trainee@redhat9 ~]$ nl .bash_history | tail
    59 ls
    60 ls | sort
    61 ls | sort -r
    62 more /etc/services
    63 less /etc/services
    64 find acc
    65 find aac
    66 su -
    67 sleep 10
    68 su -
```

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

1.6 - Generating file name endings

The /bin/bash shell is used to generate file name endings. This is accomplished by using the **[Tab]** key. In the following example, the command entered is :

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

```
[trainee@redhat9 ~]$ ls .bash
.bash_history .bash_logout .bash_profile .bashrc
```

> **Important**: Note that by pressing the Tab key three times the shell offers 4 filename completion possibilities. This is because, without more information, the shell does not know which file is involved.

The same possibility exists for generating command name endings. In this case enter the following command:

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

Press the Tab key twice. You will get a window similar to this one:

```
[trainee@redhat9 ~]$ mo
modinfo             modulemd-validator  more                mount.composefs     mount.fuse3
modprobe            monitor-sensor      mount               mount.fuse          mountpoint
```

1.7 - The interactive shell

When using the shell, we often need to execute a command on several files instead of processing them individually. For this purpose we can use 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

Metacharacter	Description
?(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 ...

The * Metacharacter

In your home directory, create a **training** directory. Then create 5 files in this directory named f1, f2, f3, f4 and f5 respectively:

```
[trainee@redhat9 ~]$ mkdir training
[trainee@redhat9 ~]$ cd training
[trainee@redhat9 training]$ touch f1 f2 f3 f4 f5
[trainee@redhat9 training]$ ls
f1 f2 f3 f4 f5
```

To demonstrate the use of the metacharacter *, enter the following command:

```
[trainee@redhat9 training]$ echo f*
f1 f2 f3 f4 f5
```

Important: Note that the * character replaces a character or a string of characters.

The ? Metacharacter

Now create the f52 and f62 files:

```
[trainee@redhat9 training]$ touch f52 f62
```

Then enter the following command:

```
[trainee@redhat9 training]$ echo f?2
f52 f62
```

Important: Note that the **?** character replaces **a single** character.

The [] Metacharacter

Usage can take several different forms:

Metacharacter	Description
[xyz]	Represents either x or y or z
[m-t]	Represents a character in the range m to 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 characters **[** and **]**, create the file a100 :

```
[trainee@redhat9 training]$ touch a100
```

Then enter the following commands and note the result:

```
[trainee@redhat9 training]$ echo [a-f]*
a100 f1 f2 f3 f4 f5 f52 f62
```

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

```
[trainee@redhat9 training]$ echo [af]*
a100 f1 f2 f3 f4 f5 f52 f62
```

Important: Note here that all files starting with the letters **a** or **f** are displayed on the screen.

```
[trainee@redhat9 training]$ echo [!a]*
f1 f2 f3 f4 f5 f52 f62
```

Important: Note here that all files are displayed on the screen, with the exception of one file beginning with the letter **a** .

```
[trainee@redhat9 training]$ echo [a-b]*
a100
```

Important: Note here that only the file starting with the letter **a** is displayed on the screen as there are no files starting with the letter **b**.

```
[trainee@redhat9 training]$ echo [a-f]
```

```
[a-f]
```

> **Important**: Note that in this case, there are no files named **a**, **b**, **c**, **d**, **e** or **f**. For this reason, having found no correspondence between the filter used and the objects in the current directory, the **echo** command returns the filter passed as an argument, i.e. **[a-f]**.

1.8 - The extglob option

Enable the **extglob** option in the bash shell so that you can use **?(expression), *(expression), +(expression), @(expression) and !(expression)**:

```
[trainee@redhat9 training]$ shopt -s extglob
```

The **shopt** command is used to enable or disable options for optional shell behaviour. The list of options can be viewed by executing the **shopt** command without options:

```
[trainee@redhat9 training]$ shopt
autocd off
assoc_expand_once off
cdable_vars off
cdspell off
checkhash off
checkjobs off
checkwinsize on
cmdhist on
compat31 off
compat32 off
compat40 off
compat41 off
compat42 off
compat43 off
```

```
compat44 off
complete_fullquote on
direxpand off
dirspell off
dotglob off
execfail off
expand_aliases on
extdebug off
extglob on
extquote on
failglob off
force_fignore on
globasciiranges on
globstar off
gnu_errfmt off
histappend on
histreedit off
histverify off
hostcomplete off
huponexit off
inherit_errexit off
interactive_comments on
lastpipe off
lithist off
localvar_inherit off
localvar_unset off
login_shell on
mailwarn off
no_empty_cmd_completion off
nocaseglob off
nocasematch off
nullglob off
progcomp on
progcomp_alias off
```

```
promptvars on
restricted_shell off
shift_verbose off
sourcepath on
syslog_history off
xpg_echo off
```

?(expression)

Create the files f, f.txt, f123.txt, f123123.txt, f123123123.txt :

```
[trainee@redhat9 training]$ touch f f.txt f123.txt f123123.txt f123123123.txt
```

Enter the following command:

```
[trainee@redhat9 training]$ ls f?(123).txt
f123.txt f.txt
```

> **Important**: Note here that the command displays files with names containing 0 or 1 occurrences of the string **123**.

***(expression)**

Enter the following command:

```
[trainee@redhat9 training]$ ls f*(123).txt
f123123123.txt f123123.txt f123123.txt f.txt
```

Important: Note here that the command displays files with names containing from 0 up to x occurrences of the string **123**.

+(expression)

Enter the following command:

```
[trainee@redhat9 training]$ ls f+(123).txt
f123123123.txt f123123.txt f123.txt
```

Important : Note here that the command displays files with names containing between 1 and x occurrences of the string **123**.

@(expression)

Enter the following command:

```
[trainee@redhat9 training]$ ls f@(123).txt
f123.txt
```

Important: Note here that the command displays files with names containing 1 single occurrence of the string **123**.

!(expression)

Enter the following command:

```
[trainee@redhat9 training]$ ls f!(123).txt
f123123123.txt f123123.txt f.txt
```

> **Important**: Note here that the command only displays files with a name that does **not** contain the string **123**.

Protecting Metacharacters

In order to use a metacharacter in a literal context, an escape character must be used. There are three escape characters:

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

To illustrate the use of escape characters, consider the following command:

```
echo * is a metacharacter [Enter].
```

When you enter this command in your **training** directory, you will get a window similar to this one:

```
[trainee@redhat9 training]$ echo * is a metacharacter
a100 f f1 f123123123.txt f123123.txt f123123.txt f2 f3 f4 f5 f52 f62 f.txt is a metacharacter

[trainee@redhat9 training]$ echo \* is a metacharacter
```

```
* is a metacharacter

[trainee@redhat9 training]$ echo "* is a metacharacter"
* is a metacharacter

[trainee@redhat9 training]$ echo ‘* is a metacharacter’
* is a metacharacter
```

1.9 - Exit codes

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

For example:

```
[trainee@redhat9 training]$ cd ..
[trainee@redhat9 ~]$ mkdir codes
[trainee@redhat9 ~]$ echo $?
0
[trainee@redhat9 ~]$ touch codes/exit.txt
[trainee@redhat9 ~]$ rmdir codes
rmdir: failed to remove ‘codes’: Directory not empty
[trainee@redhat9 ~]$ echo $?
1
```

In this example the **codes** directory was created successfully. The exit code stored in the $? variable is a zero.

Deleting the directory encountered an error because **codes** contained the file **return**. The exit code stored in the $? variable is **one**.

If the exit code is **zero**, the last command was executed without error.

If the exit code is **other than zero**, the last command was completed with an error.

1.10 - Redirections

Your dialogue with the Linux system uses input and output channels. The keyboard is called the **standard input channel** and the screen is called the **standard output channel**:

In other words, when you type a command on the keyboard, you see the result of that command on the screen.

Sometimes, however, it is useful to redirect the standard output channel to a file. In this way, the result of a command such as **free** can be stored in a file for future reference:

This effect is achieved by using a **redirection**:

```
[trainee@redhat9 ~]$ pwd
/home/trainee
[trainee@redhat9 ~]$ cd training
[trainee@redhat9 training]$ free > file
[trainee@redhat9 training]$ cat file
               total        used        free      shared  buff/cache   available
Mem:         7869560      996400     4964048       15324     2229600     6873160
Swap:        5242876           0     5242876
```

If the target file does not exist, it is created and its contents will be the result of the free command.

On the other hand if the file already exists, it will be overwritten :

```
[trainee@redhat9 training]$ date > file
[trainee@redhat9 training]$ cat file
Thu Sep 26 12:49:11 PM CEST 2024
```

To add additional data to the same target file, a **double redirect** must be used:

```
[trainee@redhat9 training]$ free >> file
[trainee@redhat9 training]$ free >> file
[trainee@redhat9 training]$ cat file
Thu Sep 26 12:49:11 PM CEST 2024
               total        used        free      shared  buff/cache   available
Mem:         7869560      996392     4964048       15324     2229608     6873168
Swap:        5242876           0     5242876
```

This way, the output of the free command will be added to the end of your file after the information in the free command.

Important: Note that the standard output can only be redirected in **one direction**.

Input and output channels are numbered:

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

The following command will create a file named **errorlog** that contains error messages from the execution of the **rmdir** command:

```
[trainee@redhat9 training]$ cd ..
[trainee@redhat9 ~]$ rmdir training/ 2>errorlog
[trainee@redhat9 ~]$ cat errorlog
rmdir: failed to remove ‘training/’: Directory not empty
```

In fact the error is generated because the **training** directory is not empty.

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

```
[trainee@redhat9 ~]$ free > file 2>&1
```

The syntax **2>&1** sends the output of channel 2 to the same place as channel 1, namely the file named **file**.

It is possible to modify the standard input channel in order to read information from a file. In this case the redirection is obtained by using the **<** character:

```
[trainee@redhat9 ~]$ wc -w < errorlog
8
```

In this example the wc command counts the number of words (-w) in the errorlog file and displays it on the screen :

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.

1.11 - Pipes

It is also possible to link commands using a | pipe.

In this case, the output channel of the command to the left of the pipe is sent to the input channel of the command to the right of the pipe :

```
[trainee@redhat9 ~]$ ls | wc -w
17
```

This command, run in your home directory, takes the output of the **ls** command and asks the **wc** command to count the number of words included in the output of ls:

Important: Note that it is possible to link several tubes in the same command.

Remember that the standard output can only be redirected in one direction. In order to be able to redirect the standard output to a file **and** view it on screen, we need to use the **tee** command with a pipe:

```
[trainee@redhat9 ~]$ date | tee file1
Thu Sep 26 12:54:36 PM CEST 2024
[trainee@redhat9 ~]$ cat file1
Thu Sep 26 12:54:36 PM CEST 2024
```

This same technique allows us to create **two files**:

```
[trainee@redhat9 ~]$ date | tee file1 > file2
[trainee@redhat9 ~]$ cat file1
Thu Sep 26 12:55:11 PM CEST 2024
[trainee@redhat9 ~]$ cat file2
Thu Sep 26 12:55:11 PM CEST 2024
```

Important: By default the tee command overwrites the destination file. To add additional data to the same target file, the **-a** option to the tee command should be used.

1.12 - Command substitutions

It is sometimes interesting, particularly in scripts, to replace a command with its output value. To illustrate this point, let's consider the following commands:

```
[trainee@redhat9 ~]$ echo date
date

[trainee@redhat9 ~]$ echo $(date)
Thu Sep 26 12:56:02 PM CEST 2024
```

```
[trainee@redhat9 ~]$ echo `date`
Thu Sep 26 12:56:17 PM CEST 2024
```

> **Important**: Note the format of each **$(command)** or **`commande`** substitution. On a French keyboard, the anti-coast is accessed using the Alt Gr and 77 keys.

1.13 - Command chaining

It is possible to group commands together using a sub-shell :

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

This example sends the results of the three commands to the **list** file, processing them in the background.

The commands can also be chained according to the exit code of the previous command.

&& is used to ensure that the second command is executed if the output status value is 0, i.e. there were no errors.

|| is used to ensure the reverse.

The syntax of this command is :

```
Command1 && Command2
```

In this case, Command2 is only executed if Command1 has run without error.

Or :

```
Command1 || Command2
```

In this case, Command2 is executed if Command1 has encountered an error.

1.14 - Displaying shell variables

A shell variable can be displayed using the command :

```
$ echo $VARIABLE [Input]
```

Main 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.

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

Internationalization and Localization Variables

Internationalization, also known as **i18n** because there are 18 letters between the letter **I** and the letter **n** in the word *Internationalization*, involves adapting software to parameters that vary from one region to another:

- 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 ...

The **Localization**, also called **l10n** because there are 10 letters between the letter **L** and the letter n in the word *Localization*, consists of modifying the internalisation according to a specific region.

The complete country code takes the following form: **language-PAYS.character_set**. For example, for the English language the language-PAYS values are :

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

The most important system variables containing regionalisation information are :

Variable	**Description**
LC_ALL	With a non-zero value, this takes precedence over the value of all other internationalisation variables.
LANG	Provides a default value for environment variables whose value is null or undefined.
LC_CTYPE	Determines the regional parameters for interpreting the sequence of bytes of text data in characters.

For example:

```
[trainee@redhat9 ~]$ echo $LC_ALL

[trainee@redhat9 ~]$ echo $LC_CTYPE

[trainee@redhat9 ~]$ echo $LANG
en_US.UTF-8
[trainee@redhat9 ~]$ locale
LANG=en_US.UTF-8
LC_CTYPE="en_US.UTF-8’
LC_NUMERIC="en_US.UTF-8’
LC_TIME="en_US.UTF-8’
LC_COLLATE="en_US.UTF-8’
LC_MONETARY="en_US.UTF-8
LC_MESSAGES="en_US.UTF-8’
LC_PAPER="en_US.UTF-8’
LC_NAME="en_US.UTF-8’
LC_ADDRESS="en_US.UTF-8’
LC_TELEPHONE="en_US.UTF-8’
```

```
LC_MEASUREMENT="en_US.UTF-8’
LC_IDENTIFICATION="en_US.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
$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

1.15 - The env command

The **env** command sends the values of the system variables in the environment of the user invoking it to the standard output:

```
[trainee@redhat9 ~]$ env
SHELL=/bin/bash
HISTCONTROL=ignoredups
HISTSIZE=1000
HOSTNAME=redhat9.ittraining.loc
PWD=/home/trainee
LOGNAME=trainee
XDG_SESSION_TYPE=tty
MOTD_SHOWN=pam
HOME=/home/trainee
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=01;37;41
```

```
:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*
.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip
=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.b
z2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;
31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.w
im=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bm
p=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01
;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35
:*.webm=01;35:*.webp=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.
wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;
35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=01;36:*.au=01;36:*.f
lac=01;36:*.m4a=01;36:*.mid=01;36:*.midi=01;36:*.mka=01;36:*.mp3=01;36:*.mpc=01;36:*.ogg=01;36:*.ra=01;36:*.wav=0
1;36:*.oga=01;36:*.opus=01;36:*.spx=01;36:*.xspf=01;36:
SSH_CONNECTION=10.0.2.1 37578 10.0.2.101 22
XDG_SESSION_CLASS=user
SELINUX_ROLE_REQUESTED=
TERM=xterm-256color
LESSOPEN=||/usr/bin/lesspipe.sh %s
USER=trainee
SELINUX_USE_CURRENT_RANGE=
SHLVL=1
XDG_SESSION_ID=4
XDG_RUNTIME_DIR=/run/user/1000
SSH_CLIENT=10.0.2.1 37578 22
which_declare=declare -f
XDG_DATA_DIRS=/home/trainee/.local/share/flatpak/exports/share:/var/lib/flatpak/exports/share:/usr/local/share:/u
sr/share
PATH=/home/trainee/.local/bin:/home/trainee/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
SELINUX_LEVEL_REQUESTED=
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
MAIL=/var/spool/mail/trainee
SSH_TTY=/dev/pts/0
BASH_FUNC_which%%=() {  ( alias;
 eval ${which_declare} ) | /usr/bin/which --tty-only --read-alias --read-functions --show-tilde --show-dot $@
```

```
}
_=/usr/bin/env
OLDPWD=/home/trainee/training
```

The command can also be used to set a variable when executing a command. For example, to run **xterm** with the EDITOR variable set to **vi**:

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

1.16 - Bash Shell options

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

```
$ set -o [Enter]
```

For example:

```
[trainee@redhat9 ~]$ 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 activate an option it is again convenient to use the **set** command:

```
[trainee@redhat9 ~]$ set -o allexport
[trainee@redhat9 ~]$ set -o
allexport on
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
```

Note that the **allexport** option has been enabled.

To disable an option, we use the **set** command with the **+o** option:

```
$ set +o allexport [Enter]
```

```
[trainee@redhat9 ~]$ set +o allexport
[trainee@redhat9 ~]$ 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
```

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 metacharacters
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

Examples

noclobber

```
[trainee@redhat9 ~]$ set -o noclobber
[trainee@redhat9 ~]$ pwd > file
-bash: file: cannot overwrite existing file
[trainee@redhat9 ~]$ pwd >| file
```

```
[trainee@redhat9 ~]$ cat file
/home/trainee
[trainee@redhat9 ~]$ set +o noclobber
```

Important: Note that the **noclobber** option can be bypassed by using the redirect followed by the **|** character.

noglob

```
[trainee@redhat9 ~]$ set -o noglob
[trainee@redhat9 ~]$ echo *
*
[trainee@redhat9 ~]$ set +o noglob
[trainee@redhat9 ~]$ echo *
aac abc bca codes Desktop Documents Downloads errorlog file file1 file2 list Music Pictures Public Templates
training Videos vitext xyz
```

Important: Note that the effect of the metacharacter is cancelled under the influence of the **noglob** option.

nounset

```
[trainee@redhat9 ~]$ set -o nounset
[trainee@redhat9 ~]$ echo $FENESTROS
-bash: FENESTROS: unbound variable
```

```
[trainee@redhat9 ~]$ set +o nounset
[trainee@redhat9 ~]$ echo $FENESTROS

[trainee@redhat9 ~]$
```

Important: Note that the non-existent variable **$FENESTROS** is identified as such under the influence of the **nounset** option. Now the usual behavior of Linux is to return an empty line which does not indicate whether the variable does not exist or whether it is simply empty.
