Dernière mise-à-jour : 2017/02/19 14:31

LRF105 - La Ligne de Commande

Le Shell

Un shell est un **interpréteur de commandes** ou en anglais un **Command Line Interpreter (C.L.I)**. Il est utilisé comme interface pour donner des instructions ou **commandes** au système d'exploitation.

Le mot shell est générique. Il existe de nombreux shells dans le monde Unix, par exemple :

Shell	Nom	Date de Sortie	Inventeur	Commande	Commentaires
tsh	Thompson Shell	1971	Ken Thompson	sh	Le premier shell
sh	Bourne Shell	1977	Stephen Bourne	sh	Le shell commun à tous les Unix. Sous RHEL/CentOS 7 : /usr/bin/sh
csh	C-Shell	1978	Bill Joy	csh	Le shell BSD. Sous RHEL/CentOS 7 : /usr/bin/csh
tcsh	Tenex C-Shell	1979	Ken Greer	tcsh	Un dérivé du shell csh. Sous RHEL/CentOS 7 : /usr/bin/tcsh
ksh	Korn Shell	1980	David Korn	ksh	Uniquement libre depuis 2005. Sous RHEL/CentOS 7 : /usr/bin/ksh
bash	Bourne Again Shell	1987	Brian Fox	bash	Le shell par défaut de Linux et de MacOS X. Sous RHEL/CentOS 7 : /usr/bin/bash
zsh	Z Shell	1990	Paul Falstad	zsh	Zsh est plutôt orienté pour l'interactivité avec l'utilisateur. Sous RHEL/CentOS 7 : /usr/bin/zsh

Sous RHEL/CentOS 7 le shell **/bin/sh** est un lien symbolique vers **/bin/bash** :

```
[trainee@centos7 ~]$ ls -l /bin/sh
lrwxrwxrwx. 1 root root 4 30 sept. 06:01 /bin/sh -> bash
```

Le Shell /bin/bash

Ce module concerne l'utilisation du shell **bash** sous Linux. Le shell **bash** permet de:

- Rappeler des commandes
- Générer la fin de noms de fichiers
- Utiliser des alias
- Utiliser les variables tableaux
- Utiliser les variables numériques et l'arithmétique du langage C
- Gérer des chaînes de caractères
- Utiliser les fonctions

Une commande commence toujours par un mot clef. Ce mot clef est interprété par le shell selon le type de commande et dans l'ordre qui suit :

1. Les alias
2. Les fonctions
3. Les commandes internes au shell
4. Les commandes externes au shell

Les Commandes Internes et Externes au shell

Les commandes internes au shell sont des commandes telles **cd**. Pour vérifier le type de commande, il faut utiliser la commande **type** :

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

Les commandes externes au shell sont des binaires exécutables ou des scripts, généralement situés dans /bin, /sbin, /usr/bin ou /usr/sbin :

```
[trainee@centos7 ~]$ type passwd
passwd is /usr/bin/passwd
```

Les alias

Les alias sont des noms permettant de désigner une commande ou une suite de commandes et ne sont spécifiques qu'au shell qui les a créés ainsi qu'à l'environnement de l'utilisateur :

```
[trainee@centos7 ~]$ type ls
ls is aliased to `ls --color=auto'
```

Important : Notez que dans ce cas l'alias **ls** est en effet un alias qui utilise la **commande** ls elle-même.

Un alias se définit en utilisant la commande **alias** :

```
[trainee@centos7 ~]$ alias dir='ls -l'
[trainee@centos7 ~]$ dir
total 4
-rw-rw-r--. 1 trainee trainee   0 29 sept. 18:20 aac
-rw-rw-r--. 1 trainee trainee   0 29 sept. 18:20 abc
-rw-rw-r--. 1 trainee trainee   0 29 sept. 18:20 bca
drwxr-xr-x. 2 trainee trainee   6 30 avril 11:54 Desktop
drwxr-xr-x. 2 trainee trainee   6 30 avril 11:54 Documents
drwxr-xr-x. 2 trainee trainee   6 30 avril 11:54 Downloads
drwxr-xr-x. 2 trainee trainee   6 30 avril 11:54 Music
drwxr-xr-x. 2 trainee trainee   6 30 avril 11:54 Pictures
drwxr-xr-x. 2 trainee trainee   6 30 avril 11:54 Public
drwxr-xr-x. 2 trainee trainee   6 30 avril 11:54 Templates
drwxr-xr-x. 2 trainee trainee   6 30 avril 11:54 Videos
-rw-rw-r--. 1 trainee trainee 442 29 sept. 00:53 vitext
-rw-rw-r--. 1 trainee trainee   0 29 sept. 18:20 xyz
```

Important : Notez que la commande **dir** existe vraiment. Le fait de créer un alias qui s'appelle **dir** implique que l'alias sera exécuté à la place de la commande **dir**.

La liste des alias définis peut être visualisée en utilisant la commande **alias** :

```
[trainee@centos7 ~]$ 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 | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
```

Important : Notez que cette liste contient, sans distinction, les alias définis dans les fichiers de démarrage du système ainsi que l'alias **dir** créé par **trainee** qui n'est que disponible à **trainee** dans le terminal courant.

Pour forcer l'exécution d'une commande et non l'alias il faut faire précéder la commande par le caractère **** :

```
[trainee@centos7 ~]$ \dir
aac  bca      Documents  Music     Public     Videos  xyz
abc  Desktop  Downloads  Pictures  Templates  vitext
```

Pour supprimer un alias, il convient d'utiliser la commande **unalias** :

```
[trainee@centos7 ~]$ unalias dir
```

```
[trainee@centos7 ~]$ dir
aac  bca      Documents  Music     Public     Videos  xyz
abc  Desktop  Downloads  Pictures  Templates  vitext
```

Le shell des utilisateurs est défini par **root** dans le dernier champs du fichier **/etc/passwd** :

```
[trainee@centos7 ~]$ 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:99:99:Nobody:/:/sbin/nologin
avahi-autoipd:x:170:170:Avahi IPv4LL Stack:/var/lib/avahi-autoipd:/sbin/nologin
systemd-bus-proxy:x:999:997:systemd Bus Proxy:/:/sbin/nologin
systemd-network:x:998:996:systemd Network Management:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:997:995:User for polkitd:/:/sbin/nologin
abrt:x:173:173::/etc/abrt:/sbin/nologin
usbmuxd:x:113:113:usbmuxd user:/:/sbin/nologin
colord:x:996:993:User for colord:/var/lib/colord:/sbin/nologin
libstoragemgmt:x:995:992:daemon account for libstoragemgmt:/var/run/lsm:/sbin/nologin
setroubleshoot:x:994:991::/var/lib/setroubleshoot:/sbin/nologin
rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin
rtkit:x:172:172:RealtimeKit:/proc:/sbin/nologin
chrony:x:993:990::/var/lib/chrony:/sbin/nologin
unbound:x:992:989:Unbound DNS resolver:/etc/unbound:/sbin/nologin
```

```
tss:x:59:59:Account used by the trousers package to sandbox the tcsd daemon:/dev/null:/sbin/nologin
geoclue:x:991:988:User for geoclue:/var/lib/geoclue:/sbin/nologin
ntp:x:38:38::/etc/ntp:/sbin/nologin
sssd:x:990:987:User for sssd:/:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
pulse:x:171:171:PulseAudio System Daemon:/var/run/pulse:/sbin/nologin
gdm:x:42:42::/var/lib/gdm:/sbin/nologin
gnome-initial-setup:x:989:984::/run/gnome-initial-setup/:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
trainee:x:1000:1000:trainee:/home/trainee:/bin/bash
vboxadd:x:988:1::/var/run/vboxadd:/bin/false
named:x:25:25:Named:/var/named:/sbin/nologin
```

Cependant l'utilisateur peut changer son shell grâce à la commande **chsh**. Les shells disponibles aux utilisateurs du système sont inscrits dans le fichier **/etc/shells**. Saisissez la commande **cat /etc/shells** :

```
[trainee@centos7 ~]$ cat /etc/shells
/bin/sh
/bin/bash
/sbin/nologin
/usr/bin/sh
/usr/bin/bash
/usr/sbin/nologin
/bin/tcsh
/bin/csh
```

Ensuite utilisez la commande **echo** pour afficher le shell actuel de **trainee** :

```
[trainee@centos7 ~]$ echo $SHELL
```

```
/bin/bash
```

Important : Notez sous RHEL/CentOS 7 que le système nous informe que le shell courant de l'utiisateur **trainee** est **/bin/bash** et non **/usr/bin/bash**. Ceci est du au fait que le répertoire /bin est un lien symbolique pointant vers le répertoire /usr/bin.

Changez ensuite le shell de **trainee** en utilisant la commande **chsh** en indiquant la valeur de **/bin/sh** pour le nouveau shell :

```
[trainee@centos7 ~]$ chsh
Changing shell for trainee.
New shell [/bin/bash]: /bin/sh
Password: trainee
Shell changed.
```

Important : Notez que le mot de passe saisi ne sera **pas** visible.

Vérifiez ensuite le shell actif pour **trainee** :

```
[trainee@centos7 ~]$ echo $SHELL
/bin/bash
```

Dernièrement contrôlez le shell stipulé dans le fichier **/etc/passwd** pour **trainee** :

```
[trainee@centos7 ~]$ cat /etc/passwd | grep trainee
trainee:x:1000:1000:trainee:/home/trainee:/bin/sh
```

Important : Vous noterez que le shell actif est toujours **/bin/bash** tandis que le shell stipulé dans le fichier /etc/passwd est le **/bin/sh**. Le shell

/bin/sh ne deviendra le shell actif de **trainee** que lors de sa prochaine connexion au système.

Modifiez votre shell à **/bin/bash** de nouveau en utilisant la commande chsh :

```
[trainee@centos7 ~]$ chsh
Changing shell for trainee.
New shell [/bin/sh]: /bin/bash
Password: trainee
Shell changed.
```

Important : Notez que le mot de passe saisi ne sera **pas** visible.

Le Prompt

Le prompt d'un utilisateur dépend de son statut :

- **$** pour un utilisateur normal,
- **#** pour root.

Rappeler des Commandes

Le shell **/bin/bash** permet le rappel des dernières commandes saisies. Afin de connaître la liste des commandes mémorisées, utilisez la commande history :

```
[trainee@centos7 ~]$ history | more
    1  su -
    2  df -h
```

```
    3  su -
    4  exit
    5  su -
    6  su -
    7  vi vitext
    8  view vitext
    9  vi vitext
   10  locale
   11  LANG=en_GB.UTF-8
   12  export LANG
   13  locale
   14  vi vitext
   15  vi .exrc
   16  vi vitext
   17  clear
   18  stty -a
   19  date
   20  locale
   21  who
   22  df
   23  df -h
--More--
```

Important: L'historique est spécifique à chaque utilisateur.

L'historique des commandes est en mode **emacs** par défaut. De ce fait, le rappel de la dernière commande se fait en utilisant la touche **[Flèche vers le haut]** ou bien les touches **[CTRL]-[P]** et le rappel de la commande suivante se fait en utilisant la touche **[Flèche vers le bas]** ou bien les touches **[CTRL]-[N]** :

Caractère de Contrôle	**Définition**
[CTRL]-[P] (= flèche vers le haut)	Rappelle la commande précédente

Caractère de Contrôle	**Définition**
[CTRL]-[N] (= flèche vers le bas)	Rappelle la commande suivante

Pour se déplacer dans la ligne de l'historique :

Caractère de Contrôle	**Définition**
[CTRL]-[A]	Se déplacer au début de la ligne
[CTRL]-[E]	Se déplacer à la fin de la ligne
[CTRL]-[B]	Se déplacer un caractère à gauche
[CTRL]-[F]	Se déplacer un caractère à droite
[CTRL]-[D]	Supprimer le caractère sous le curseur

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

Caractère de Contrôle	**Définition**
[CTRL]-[R] *chaine*	Recherche en arrière de *chaine* dans l'historique. L'utilisation successive de la combinaison de touches par la suite recherche d'autres occurences de *chaine*
[CTRL]-[S] *chaine*	Recherche en avant de *chaine* dans l'historique. L'utilisation successive de la combinaison de touches par la suite recherche d'autres occurences de *chaine*
[CTRL]-[G]	Sortir du mode recherche

Il est aussi possible de rappeler la dernière commande de l'historique en utilisant les caractères **!!**:

```
[trainee@centos7 ~]$ ls
aac  bca      Documents  Music     Public     Videos  xyz
abc  Desktop  Downloads  Pictures  Templates  vitext
[trainee@centos7 ~]$ !!
ls
aac  bca      Documents  Music     Public     Videos  xyz
abc  Desktop  Downloads  Pictures  Templates  vitext
```

Vous pouvez rappeler une commande spécifique de l'historique en utilisant le caractère **!** suivi du numéro de la commande à rappeler :

```
[trainee@centos7 ~]$ !123
ls
aac  bca      Documents  Music     Public     Videos  xyz
abc  Desktop  Downloads  Pictures  Templates  vitext
```

Le paramétrage de la fonction du rappel des commandes est fait pour tous les utilisateurs dans le fichier **/etc/profile**. Dans ce fichier, les variables concernant le rappel des commandes peuvent être définis. Le plus important est **HISTSIZE** :

```
[trainee@centos7 ~]$ cat /etc/profile | grep HISTSIZE
HISTSIZE=1000
export PATH USER LOGNAME MAIL HOSTNAME HISTSIZE HISTCONTROL
```

Vous noterez que dans le cas précédent, la valeur de **HISTSIZE** est de **1000**. Ceci implique que les dernières mille commandes sont mémorisées.

Les commandes mémorisées sont stockées dans le fichier **~/.bash_history**. Les commandes de la session en cours ne sont sauvegardées dans ce fichier qu'à la fermerture de la session :

```
[trainee@centos7 ~]$ nl .bash_history | more
     1  su -
     2  df -h
     3  su -
     4  exit
     5  su -
     6  su -
     7  vi vitext
     8  view vitext
     9  vi vitext
    10  locale
    11  LANG=en_GB.UTF-8
    12  export LANG
    13  locale
    14  vi vitext
    15  vi .exrc
    16  vi vitext
```

```
    17  clear
    18  stty -a
    19  date
    20  locale
    21  who
    22  df
    23  df -h
--More--
```

Important : Notez l'utilisation de la commande **nl** pour numéroter les lignes de l'affichage du contenu du fichier **.bash_history**.

Générer les fins de noms de fichiers

Le shell /bin/bash permet la génération des fins de noms de fichiers. Celle-ci est accomplie grâce à l'utilisation de la touche **[Tab]**. Dans l'exemple qui suit, la commande saisie est :

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

```
[trainee@centos7 ~]$ ls .bash
.bash_history  .bash_logout   .bash_profile  .bashrc
```

Important : Notez qu'en appuyant sur la touche Tab trois fois le shell propose 3 ou 4 possibilités de complétion de nom de fichier. En effet, sans plus d'information, le shell ne sait pas quel fichier est concerné.

La même possibilité existe pour la génération des fins de noms de commandes. Dans ce cas saisissez la commande suivante :

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

Appuyez sur la touche Tab deux fois. Vous obtiendrez une fenêtre similaire à celle-ci :

```
[trainee@centos7 ~]$ mo
mobj_dump        modutil          mount.cifs       mount.nfs4       mousetweaks
modifyrepo       mokutil          mount.fuse       mountpoint
modinfo          more             mount.glusterfs  mountstats
modprobe         mount            mount.nfs        mount.vboxsf
```

Le shell interactif

Lors de l'utilisation du shell, nous avons souvent besoin d'exécuter une commande sur plusieurs fichiers au lieu de les traiter individuellement. A cette fin nous pouvons utiliser les caractères spéciaux.

Caractère Spéciaux	Description
*	Représente 0 ou plus de caractères
?	Représente un caractère
[abc]	Représente un caractère parmi ceux entre crochets
[!abc]	Représente un caractère ne trouvant pas parmi ceux entre crochets
?(expression1|expression2| ...)	Représente 0 ou 1 fois l'expression1 ou 0 ou 1 fois l'expression2 ...
***(expression1|expression2| ...)**	Représente 0 à x fois l'expression1 ou 0 à x fois l'expression2 ...
+(expression1|expression2| ...)	Représente 1 à x fois l'expression1 ou 1 à x fois l'expression2 ...
@(expression1|expression2| ...)	Représente 1 fois l'expression1 ou 1 fois l'expression2 ...
!(expression1|expression2| ...)	Représente 0 fois l'expression1 ou 0 fois l'expression2 ...

Caractère *

Dans votre répertoire individuel, créez un répertoire **training**. Ensuite créez dans ce répertoire 5 fichiers nommés respectivement f1, f2, f3, f4 et f5 :

```
[trainee@centos7 ~]$ mkdir training
```

```
[trainee@centos7 ~]$ cd training
[trainee@centos7 training]$ touch f1 f2 f3 f4 f5
```

Afin de démontrer l'utilisation du caractère spécial *, saisissez la commande suivante :

```
[trainee@centos7 training]$ echo f*
f1 f2 f3 f4 f5
```

Important : Notez que le caractère * remplace un caractère ou une suite de caractères.

Caractère ?

Créez maintenant les fichiers f52 et f62 :

```
[trainee@centos7 training]$ touch f52 f62
```

Saisissez ensuite la commande suivante :

```
[trainee@centos7 training]$ echo f?2
f52 f62
```

Important : Notez que le caractère **?** remplace **un seul** caractère.

Caractères [ ]

L'utilisation peut prendre plusieurs formes différentes :

Joker	Description
[xyz]	Représente le caractère x ou y ou z
[m-t]	Représente le caractère m ou n .... t
[!xyz]	Représente un caractère autre que x ou y ou z
[!m-t]	Représente un caractère autre que m ou n .... t

Afin de démontrer l'utilisation des caractères **[** et **]**, créez le fichier a100 :

```
[trainee@centos7 training]$ touch a100
```

Ensuite saisissez les commandes suivantes et notez le résultat :

```
[trainee@centos7 training]$ echo [a-f]*
a100 f1 f2 f3 f4 f5 f52 f62
[trainee@centos7 training]$ echo [af]*
a100 f1 f2 f3 f4 f5 f52 f62
```

Important : Notez ici que tous les fichiers commençant par les lettres **a**, **b**, **c**, **d**, **e** ou **f** sont affichés à l'écran.

```
[trainee@centos7 training]$ echo [!a]*
f1 f2 f3 f4 f5 f52 f62
```

Important : Notez ici que tous les fichiers sont affichés à l'écran, à l'exception d'un fichier commençant par la lettre **a** .

```
[trainee@centos7 training]$ echo [a-b]*
a100
```

Important : Notez ici que seul le fichier commençant par la lettre **a** est affiché à l'écran car il n'existe pas de fichiers commençant par la lettre **b**.

```
[trainee@centos7 training]$ echo [a-f]
[a-f]
```

Important : Notez que dans ce cas, il n'existe pas de fichiers dénommés **a**, **b**, **c**, **d**, **e** ou **f**. Pour cette raison, n'ayant trouvé aucune correspondance entre le filtre utilisé et les objets dans le répertoire courant, le commande **echo** retourne le filtre passé en argument, c'est-à-dire **[a-f]**.

L'option extglob

Activez l'option **extglob** du shell bash afin de pouvoir utiliser **?(expression), *(expression), +(expression), @(expression) et !(expression)** :

```
[trainee@centos7 training]$ shopt -s extglob
```

La commande **shopt** est utilisée pour activer ou désactiver les options du comportement optional du shell. La liste des options peut être visualisée en exécutant la commande **shopt** sans options :

```
[trainee@centos7 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)

Créez les fichiers f, f.txt, f123.txt, f123123.txt, f123123123.txt :

```
[trainee@centos7 training]$ touch f f.txt f123.txt f123123.txt f123123123.txt
```

Saisissez la commande suivante :

```
[trainee@centos7 training]$ ls f?(123).txt
f123.txt  f.txt
```

Important : Notez ici que la commande affiche les fichiers ayant un nom contenant 0 ou 1 occurence de la chaîne **123**.

***(expression)**

Saisissez la commande suivante :

```
[trainee@centos7 training]$ ls f*(123).txt
f123123123.txt  f123123.txt  f123.txt  f.txt
```

Important : Notez ici que la commande affiche les fichiers ayant un nom contenant de 0 jusqu'à x occurences de la chaîne **123**.

+(expression)

Saisissez la commande suivante :

```
[trainee@centos7 training]$ ls f+(123).txt
f123123123.txt  f123123.txt  f123.txt
```

Important : Notez ici que la commande affiche les fichiers ayant un nom contenant entre 1 et x occurences de la chaîne **123**.

@(expression)

Saisissez la commande suivante :

```
[trainee@centos7 training]$ ls f@(123).txt
f123.txt
```

Important : Notez ici que la commande affiche les fichiers ayant un nom contenant 1 seule occurence de la chaîne **123**.

!(expression)

Saisissez la commande suivante :

```
[trainee@centos7 training]$ ls f!(123).txt
f123123123.txt  f123123.txt  f.txt
```

Important : Notez ici que la commande n'affiche que les fichiers ayant un nom qui ne contient **pas** la chaîne **123**.

Caractères d'Échappement

Afin d'utiliser un caractère spécial dans un contexte littéral, il faut utiliser un caractère d'échappement. Il existe trois caractères d'échappement :

Caractère	Description
\	Protège le caractère qui le suit
' '	Protège tout caractère, à l'exception du caractère ' lui-même, se trouvant entre les deux '
“ ”	Protège tout caractère, à l'exception des caractères **“** lui-même, **$**, \ et ', se trouvant entre les deux **“**

Afin d'illustrer l'utilisation des caractères d'échappement, considérons la commande suivante :

```
$ echo * est un caractère spécial [Entrée]
```

Lors de la saisie de cette commande dans votre répertoire **training**, vous obtiendrez une fenêtre similaire à celle-ci :

```
[trainee@centos7 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@centos7 training]$ echo \* est un caractère spécial
* est un caractère spécial

[trainee@centos7 training]$ echo "* est un caractère spécial"
```

```
* est un caractère spécial

[trainee@centos7 training]$ echo '* est un caractère spécial'
* est un caractère spécial
```

Codes Retour

Chaque commande retourne un code à la fin de son exécution. La variable spéciale **$?** sert à stocker le code retour de la dernière commande exécutée.

Par exemple :

```
[trainee@centos7 training]$ cd ..
[trainee@centos7 ~]$ mkdir codes
[trainee@centos7 ~]$ echo $?

[trainee@centos7 ~]$ touch codes/exit.txt
[trainee@centos7 ~]$ rmdir codes
rmdir: failed to remove ‘codes’: Directory not empty
[trainee@centos7 ~]$ echo $?
1
```

Dans cette exemple la création du répertoire **codes** s'est bien déroulée. Le code retour stocké dans la variable $? est un zéro.

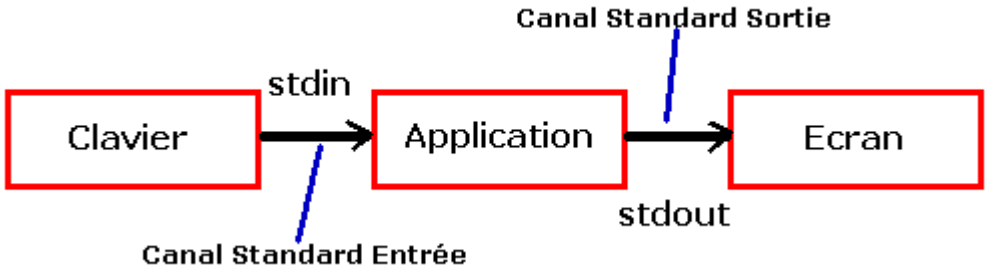
La suppression du répertoire a rencontré une erreur car **codes** contenait le fichier **retour**. Le code retour stocké dans la variable $? est un **un**.

Si le code retour est **zéro** la dernière commande s'est déroulée sans erreur.

Si le code retour est **autre que zéro** la dernière commande s'est déroulée avec une erreur.

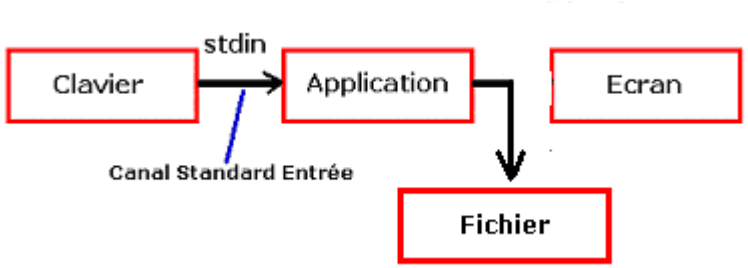
Redirections

Votre dialogue avec le système Linux utilise des canaux d'entrée et de sortie. On appelle le clavier, le **canal d'entrée standard** et l'écran, le **canal de sortie standard** :



Autrement dit, en tapant une commande sur le clavier, vous voyez le résultat de cette commande à l'écran.

Parfois, cependant il est utile de re-diriger le canal de sortie standard vers un fichier. De cette façon, le résultat d'une commande telle **free** peut être stocké dans un fichier pour une consultation ultérieure :



Cet effet est obtenu en utilisant une **redirection** :

```
[trainee@centos7 ~]$ pwd
/home/trainee
[trainee@centos7 ~]$ cd training
[trainee@centos7 training]$ free > file
[trainee@centos7 training]$ cat file
              total        used        free      shared  buff/cache   available
Mem:         500780      192692       38916        4824      269172      260472
```

```
Swap:       2096124           0     2096124
```

Si le fichier cible n'existe pas, il est créé et son contenu sera le résultat de la commande free.

Par contre si le fichier existe déjà, il sera écrasé :

```
[trainee@centos7 training]$ date > file
[trainee@centos7 training]$ cat file
Mon 28 Nov 15:48:09 CET 2016
```

Pour ajouter des données supplémentaires au même fichier cible, il faut utiliser une **double redirection** :

```
[trainee@centos7 training]$ free >> file
[trainee@centos7 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
```

De cette façon, la date du jour sera rajoutée à la fin de votre fichier après les informations de la commande free.

Important : Notez que la sortie standard ne peut être redirigée que dans **une seule direction**.

Les canaux d'entrées et de sorties sont numérotés :

- 0 = Le Canal d'entrée Standard
- 1 = Le Canal de Sortie Standard
- 2 = Le Canal d'erreur

La commande suivante créera un fichier nommé **errorlog** qui contient les messages d'erreur de l'exécution de la commande **rmdir** :

```
[trainee@centos7 training]$ cd ..
[trainee@centos7 ~]$ rmdir training/ 2>errorlog
[trainee@centos7 ~]$ cat erreurlog
rmdir: failed to remove ‘training/’: Directory not emptyy
```

En effet l'erreur est générée parce que le répertoire **training** n'est pas vide.

Nous pouvons également réunir des canaux. Pour mettre en application ceci, il faut comprendre que le shell traite les commandes de **gauche à droite**.

Dans l’exemple suivant, nous réunissons le canal de sortie et le canal d’erreurs :

```
[trainee@centos7 ~]$ free > file 2>&1
```

La syntaxe **2>&1** envoie la sortie du canal 2 au même endroit que le canal 1, à savoir le fichier dénommé **file**.

Il est possible de modifier le canal d'entrée standard afin de lire des informations à partir d’un fichier. Dans ce cas la redirection est obtenue en utilisant le caractère **<** :

```
$ wc -w < erreurlog [Entrée]
```

Dans cet exemple la commande wc compte le nombre de mots (-w) dans le fichier errorlog et l’affiche à l’écran :

```
[trainee@centos7 ~]$ wc -w < errorlog
8
```

D'autres redirections existent :

Caractères	**Définition**
&>	Rediriger les canaux 1 et 2 au même endroit
<<	Permet d'utiliser le texte taper ensuite en tant que entrée standard. Par exemple *programme* << EOF utilisera le texte taper après en tant qu'entrée standard jusqu'à l'apparition de EOF sur une ligne seule.
<>	Permet d'utiliser le fichier specifié en tant que entrée standard et sortie standard

Tubes

Il est aussi possible de relier des commandes avec un tube **|** .

Dans ce cas, le canal de sortie de la commande à gauche du tube est envoyé au canal d'entrée de la commande à droite du tube :

```
$ ls | wc -w [Entrée]
```

Cette commande, lancée dans votre répertoire personnel, prend la sortie de la commande **ls** et demande à la commande **wc** de compter le nombre de mots inclus dans la sortie de ls :

```
[trainee@centos7 ~]$ ls | wc -w
17
```

Important : Il est à noter qu'il est possible de relier plusieurs tubes dans la même commande.

Rappelez-vous que la sortie standard ne peut être redirigée que dans une seule direction. Afin de pouvoir rediriger la sortie standard vers un fichier **et** la visualiser à l'écran, nous devons utiliser la commande **tee** avec un pipe :

```
[trainee@centos7 ~]$ date | tee file1
Mon 28 Nov 16:14:24 CET 2016
[trainee@centos7 ~]$ cat file1
Mon 28 Nov 16:14:24 CET 2016
```

Cette même technique nous permet de créer **deux fichiers** :

```
$ date | tee fichier1 > fichier2 [Entrée]
```

```
[trainee@centos7 ~]$ date | tee fichier1 > fichier2
[trainee@centos7 ~]$ cat fichier1
```

```
Mon 28 Nov 16:15:57 CET 2016
[trainee@centos7 ~]$ cat fichier2
Mon 28 Nov 16:15:57 CET 2016
```

Important : Par défaut la commande tee écrase le fichier de destination. Pour ajouter des données supplémentaires au même fichier cible, il convient d'utiliser l'option **-a** de la commande tee.

Substitutions de Commandes

Il est parfois intéressant, notamment dans les scripts, de remplacer une commande par sa valeur de sa sortie. Afin d'illustrer ce point, considérons les commandes suivantes :

```
[trainee@centos7 ~]$ echo date
date
[trainee@centos7 ~]$ echo $(date)
Mon 28 Nov 16:19:35 CET 2016
[trainee@centos7 ~]$ echo `date`
Mon 28 Nov 16:19:35 CET 2016
```

Important : Notez le format de chaque substitution **$(commande)** ou **`commande`**. Sur un clavier français, l'anti-côte est accessible en utilisant les touches `Alt Gr` et 77.

Chainage de Commandes

Il est possible de regrouper des commandes à l’aide d’un sous-shell :

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

Cet exemple envoie le résultat des trois commandes vers le fichier **list** en les traitant en tâches de fond.

Les commandes peuvent être aussi chainées en fonction du code retour de la commande précédente.

&& est utilisé afin de s'assurer que la deuxième commande s'exécute dans le cas où la valeur du statut de sortie est 0, autrement dit qu'il n'y a pas eu d'erreurs.

|| est utilisé afin de s'assurer de l'inverse.

Le syntaxe de cette commande est :

```
Commande1 && Commande2
```

Dans ce cas, Commande 2 est exécutée uniquement dans le cas où Commande1 s'est exécuté sans erreur

Ou :

```
Commande1 || Commande2
```

Dans ce cas, Commande2 est exécuté si Commande1 a rencontré une erreur.

Affichage des variables du shell

Une variable du shell peut être affichée grâce à la commande :

```
$ echo $VARIABLE [Entrée]
```

Les variables principales

Variable	Description
BASH	Le chemin complet du shell.
BASH_VERSION	La version du shell.
EUID	EUID de l'utilisateur courant.
UID	UID de l'utilisateur courant.
PPID	Le PID du processus père.
PWD	Le répertoire courant.
OLDPWD	Le répertoire avant la dernière commande cd. Même chose que la commande **cd -**.
RANDOM	Un nombre aléatoire entre 0 et 32767
SECONDS	Le nombre de scondes écoules depuis le lancement du shell
LINES	Le nombre de lignes de l'écran.
COLUMNS	La largeur de l'écran.
HISTFILE	Le fichier historique
HISTFILESIZE	La taille du fichier historique
HISTSIZE	Le nombre de commandes mémorisées dans le fichier historique
HISTCMD	Le numéro de la commande courante dans l'historique
HISTCONTROL	**ignorespace** ou **ignoredups** ou **ignoreboth**
HOME	Le répertoire de connexion.
HOSTTYPE	Le type de machine.
OSTYPE	Le système d'exploitation.
MAIL	Le fichier contenant le courrier.
MAILCHECK	La fréquence de vérification du courrier en secondes.
PATH	Le chemin de recherche des commandes.
PROMPT_COMMAND	La commande exécutée avant chaque affichage du prompt.
PS1	Le prompt par défaut.
PS2	Le deuxième prompt par défaut
PS3	Le troisième prompt par défaut
PS4	Le quatrième prompt par défaut
SHELL	Le shell de préférence.
SHLVL	Le nombre d'instances du shell.
TMOUT	Le nombre de secondes moins 60 d'inactivité avant que le shell exécute la commande **exit**.

Les Variables de Régionalisation et d'Internationalisation

L'**Internationalisation**, aussi appelé **i18n** car il y a 18 lettres entre la lettre **I** et la lettre **n** dans le mot *Internationalization*, consiste à adapter un logiciel aux paramètres variant d'une région à l'autre :

- longueur des mots,
- accents,
- écriture de gauche à droite ou de droite à gauche,
- unité monétaire,
- styles typographiques et modèles rédactionnels,
- unités de mesures,
- affichage des dates et des heures,
- formats d'impression,
- format du clavier,
- etc ...

Le **Régionalisation**, aussi appelé **l10n** car il y a 10 lettres entre la lettre **L** et la lettre n du mot *Localisation*, consiste à modifier l'internalisation en fonction d'une région spécifique.

Le code pays complet prend la forme suivante : **langue-PAYS.jeu_de_caractères**. Par exemple, pour la langue anglaise les valeurs de langue-PAYS sont :

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

Les variables système les plus importants contenant les informations concernant le régionalisation sont :

Variable	Description
LC_ALL	Avec une valeur non nulle, celle-ci prend le dessus sur la valeur de toutes les autres variables d'internationalisation
LANG	Fournit une valeur par défaut pour les variables d'environnement dont la valeur est nulle ou non définie.

Variable	Description
LC_CTYPE	Détermine les paramètres régionaux pour l'interprétation de séquence d'octets de données texte en caractères.

Par exemple :

```
[trainee@centos7 ~]$ echo $LC_ALL
en_GB.UTF-8
[trainee@centos7 ~]$ echo $LC_CTYPE

[trainee@centos7 ~]$ echo $LANG
en_GB.UTF-8

[trainee@centos7 ~]$ 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
```

Les variables spéciales

Variable	Description
$LINENO	Contient le numéro de la ligne courante du script ou de la fonction
$$	Contient le PID du shell en cours

Variable	Description
$PPID	Contient le PID du processus parent du shell en cours
$0	Contient le nom du script en cours tel que ce nom ait été saisi sur la ligne de commande
$1, $2 ...	Contient respectivement le premier argument, deuxième argument etc passés au script
$#	Contient le nombre d'arguments passés au script
$*	Contient l'ensemble des arguments passés au script
$@	Contient l'ensemble des arguments passés au script

La Commande env

La commande **env** envoie sur la sortie standard les valeurs des variables système de l'environnement de l'utilisateur qui l'invoque :

```
[trainee@centos7 ~]$ env
XDG_SESSION_ID=1
HOSTNAME=centos7.fenestros.loc
SELINUX_ROLE_REQUESTED=
TERM=xterm-256color
SHELL=/bin/bash
HISTSIZE=1000
SSH_CLIENT=10.0.2.2 33896 22
SELINUX_USE_CURRENT_RANGE=
SSH_TTY=/dev/pts/0
LC_ALL=en_GB.UTF-8
USER=trainee
LS_COLORS=rs=0:di=38;5;27:ln=38;5;51:mh=44;38;5;15: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=05;48;5;232;38;5;15: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;34:*.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:*.zip=38;5;9:*.z=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:*.x
z=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:*.jpg=38;5;13:*.jpeg=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:*.xcf=38;5;13:*.xwd=38;5;13:*.yuv=38;5;13:*.cgm=38;5;13:*.emf=38;5;13:*.axv=38;5;13:*.a
nx=38;5;13:*.ogv=38;5;13:*.ogx=38;5;13:*.aac=38;5;45:*.au=38;5;45:*.flac=38;5;45:*.mid=38;5;45:*.midi=38;5;45:*.m
ka=38;5;45:*.mp3=38;5;45:*.mpc=38;5;45:*.ogg=38;5;45:*.ra=38;5;45:*.wav=38;5;45:*.axa=38;5;45:*.oga=38;5;45:*.spx
=38;5;45:*.xspf=38;5;45:
MAIL=/var/spool/mail/trainee
PATH=/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/trainee/.local/bin:/home/trainee/bin
PWD=/home/trainee
LANG=fr_FR.UTF-8
SELINUX_LEVEL_REQUESTED=
HISTCONTROL=ignoredups
SHLVL=1
HOME=/home/trainee
LOGNAME=trainee
SSH_CONNECTION=10.0.2.2 33896 192.168.1.99 22
LESSOPEN=||/usr/bin/lesspipe.sh %s
XDG_RUNTIME_DIR=/run/user/1000
_=/usr/bin/env
OLDPWD=/home/trainee/training
```

La commande peut aussi être utilisée pour fixer une variable lors de l'exécution d'une commande. Par exemple, pour lancer **xterm** avec la variable EDITOR fixée à **vi** :

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

Options du Shell Bash

Pour visualiser les options du shell bash, il convient d'utiliser la commande **set** :

```
$ set -o [Entrée]
```

Par exemple :

```
[trainee@centos7 ~]$ 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
```

Pour activer une option il convient de nouveau à utiliser la commande **set** :

set -o allexport [Entrée]

Par exemple :

```
[trainee@centos7 ~]$ set -o allexport
[trainee@centos7 ~]$ set -o
allexport       on
braceexpand     on
...
```

Notez que l'option **allexport** a été activée.

Pour désactiver une option, on utilise la commande **set** avec l'option **+o** :

```
$ set +o allexport [Entrée]
```

```
[trainee@centos7 ~]$ set +o allexport
[trainee@centos7 ~]$ set -o
allexport       off
braceexpand     on
...
```

Parmi les options, voici la description des plus intéressantes :

Option	Valeur par Défaut	Description
allexport	off	Le shell export automatiquement toute variable
emacs	on	L'édition de la ligne de commande est au style emacs
history	on	L'historique des commandes est activé
noclobber	off	Les simples re-directions n'écrasent pas le fichier de destination
noglob	off	Désactive l'expansion des caractères génériques
nounset	off	Le shell retourne une erreur lors de l'expansion d'une variable inconnue
verbose	off	Affiche les lignes de commandes saisies
vi	off	L'édition de la ligne de commande est au style vi

Exemples

noclobber

```
[trainee@centos7 ~]$ set -o noclobber
[trainee@centos7 ~]$ pwd > file
-bash: file: cannot overwrite existing file
[trainee@centos7 ~]$ pwd > file
-bash: file: cannot overwrite existing file
[trainee@centos7 ~]$ pwd >| file
[trainee@centos7 ~]$ set +o noclobber
```

Important : Notez que l'option **noclobber** peut être contournée en utilisant la redirection suivi par le caractère **|**.

noglob

```
[trainee@centos7 ~]$ set -o noglob
[trainee@centos7 ~]$ echo *
*
[trainee@centos7 ~]$ set +o noglob
[trainee@centos7 ~]$ echo *
aac abc bca codes Desktop Documents Downloads errorlog file file1 Music Pictures Public Templates training Videos
vitext xyz
```

Important : Notez que l'effet du caractère spécial est annulé sous l'influence de l'option **noglob**.

nounset

```
[trainee@centos7 ~]$ set -o nounset
[trainee@centos7 ~]$ echo $FENESTROS
-bash: FENESTROS: unbound variable
[trainee@centos7 ~]$ set +o nounset
[trainee@centos7 ~]$ echo $FENESTROS

[trainee@centos7 ~]$
```

Important : Notez que la variable inexistante **$FENESTROS** est identifiée comme telle sous l'influence de l'option **nounset**. Or le comportement habituel de Linux est de retourner une ligne vide qui n'indique pas si la variable n’existe pas ou si elle est simplement vide.

Les Scripts Shell

Le but de la suite de cette unité est de vous amener au point où vous êtes capable de comprendre et de déchiffrer les scripts, notamment les scripts de démarrage ainsi que les scripts de contrôle des services.

Écrire des scripts compliqués est en dehors de la portée de cette unité car il nécessite une approche programmation qui ne peut être adressée que lors d'une formation dédiée à l'écriture des scripts.

Exécution

Un script shell est un fichier dont le contenu est lu en entrée standard par le shell. Le contenu du fichier est lu et exécuté d'une manière séquentielle. Afin qu'un script soit exécuté, il suffit qu'il puisse être lu au quel cas le script est exécuté par un shell fils soit en l'appelant en argument à l'appel du shell :

/bin/bash myscript

soit en redirigeant son entrée standard :

/bin/bash < myscript

Dans le cas où le droit d'exécution est positionné sur le fichier script et à condition que celui-ci se trouve dans un répertoire spécifié dans le PATH de l'utilisateur qui le lance, le script peut être lancé en l'appelant simplement par son nom :

myscript

Dans le cas où le script doit être exécuté par le shell courant, dans les mêmes conditions que l'exemple précédent, et non par un shell fils, il convient de le lancer ainsi :

. myscript et **./myscript**

Dans un script il est fortement conseillé d'inclure des commentaires. Les commentaires permettent à d'autres personnes de comprendre le script. Toute ligne de commentaire commence avec le caractère **#**.

Il existe aussi un **pseudo commentaire** qui est placé au début du script. Ce pseudo commentaire permet de stipuler quel shell doit être utilisé pour l'exécution du script. L'exécution du script est ainsi rendu indépendant du shell de l'utilisateur qui le lance. Le pseudo commentaire commence avec les caractères **#!**. Chaque script commence donc par une ligne similaire à celle-ci :

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

Puisque un script contient des lignes de commandes qui peuvent être saisies en shell intéractif, il est souvent issu d'une procédure manuelle. Afin de faciliter la création d'un script il existe une commande, **script**, qui permet d'enregistrer les textes sortis sur la sortie standard, y compris le prompt dans un fichier dénommé **typescript**. Afin d'illustrer l'utilisation de cette commande, saisissez la suite de commandes suivante :

```
[trainee@centos7 ~]$ script
Script started, file is typescript
[trainee@centos7 ~]$ pwd
/home/trainee
[trainee@centos7 ~]$ ls
aac  bca    Desktop    Downloads  fichier1  file   Music     Public     training    Videos  xyz
```

```
abc  codes  Documents  errorlog   fichier2  file1  Pictures  Templates  typescript  vitext
[trainee@centos7 ~]$ exit
exit
Script done, file is typescript
[trainee@centos7 ~]$ cat typescript
Script started on Tue 29 Nov 2016 03:58:33 CET
[trainee@centos7 ~]$ pwd
/home/trainee
[trainee@centos7 ~]$ ls
aac  bca    Desktop    Downloads  fichier1  file   Music     Public     training    Videos  xyz
abc  codes  Documents  errorlog   fichier2  file1  Pictures  Templates  typescript  vitext
[trainee@centos7 ~]$ exit
exit

Script done on Tue 29 Nov 2016 03:58:40 CET
```

Cette procédure peut être utilisée pour enregistrer une suite de commandes longues et compliquées afin d'écrire un script.

Pour illustrer l'écriture et l'exécution d'un script, éditez le fichier **myscript** avec **vi** :

```
$ vi myscript [Entrée]
```

Éditez votre fichier ainsi :

```
pwd
ls
```

Sauvegardez votre fichier. Lancez ensuite votre script en passant le nom du fichier en argument à /bin/bash :

```
[trainee@centos7 ~]$ vi myscript
[trainee@centos7 ~]$ /bin/bash myscript
/home/trainee
aac  codes  Downloads  fichier2  myscript    Public       typescript  xyz
abc  Desktop    errorlog   file      Music    Templates  Videos
```

```
bca  Documents  fichier1   file1     Pictures    training   vitext
```

Lancez ensuite le script en redirigeant son entrée standard :

```
[trainee@centos7 ~]$ /bin/bash < myscript
/home/trainee
aac  codes  Downloads  fichier2  myscript    Public       typescript  xyz
abc  Desktop    errorlog   file      Music    Templates  Videos
bca  Documents  fichier1   file1     Pictures    training   vitext
```

Pour lancer le script en l'appelant simplement par son nom, son chemin doit être inclus dans votre PATH:

```
[trainee@centos7 ~]$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/trainee/.local/bin:/home/trainee/bin
```

Dans le cas de RHEL/CentOS, même si PATH contient $HOME/bin, le répertoire n'existe pas :

```
[trainee@centos7 ~]$ ls
aac  codes      Downloads  fichier2  myscript  Public     typescript  xyz
abc  Desktop    errorlog   file      Music      Templates  Videos
bca  Documents  fichier1   file1     Pictures   training   vitext
```

Créez donc ce répertoire :

```
[trainee@centos7 ~]$ mkdir bin
```

Ensuite déplacez votre script dans ce répertoire et rendez-le exécutable pour votre utilisateur :

```
[trainee@centos7 ~]$ mv myscript ~/bin
[trainee@centos7 ~]$ chmod u+x ~/bin/myscript
```

Exécutez maintenant votre script en l'appelant par son nom à partir du répertoire **/tmp** :

```
[trainee@centos7 tmp]$ myscript
/tmp
hsperfdata_root  systemd-private-e526abcf335b40949dfc725f28456502-cups.service-u0xGiL
```

Placez-vous dans le répertoire contenant le script et saisissez les commandes suivantes :

- ./myscript
- . myscript

```
[trainee@centos7 tmp]$ cd ~/bin
[trainee@centos7 bin]$ ./myscript
/home/trainee/bin
myscript
[trainee@centos7 bin]$ . myscript
/home/trainee/bin
myscript
```

A faire : Notez bien la différence entre les sorties de cette dernière commande et la précédente. Expliquez pourquoi.

La commande read

La commande **read** lit son entrée standard et affecte les mots saisis dans la ou les variable(s) passée(s) en argument. La séparation entre le contenu des variables est l'espace. Par conséquent il est intéressant de noter les exemples suivants :

```
[trainee@centos7 bin]$ read var1 var2 var3 var4
fenestros edu is great!
[trainee@centos7 bin]$ echo $var1
fenestros
[trainee@centos7 bin]$ echo $var2
edu
```

```
[trainee@centos7 bin]$ echo $var3
is
[trainee@centos7 bin]$ echo $var4
great!
```

Important: Notez que chaque champs a été placé dans une variable différente. Notez aussi que par convention les variables déclarées par des utilisateurs sont en miniscules afin de les distinguer des variables système qui sont en majuscules.

```
[trainee@centos7 bin]$ read var1 var2
fenestros edu is great!
[trainee@centos7 bin]$ echo $var1
fenestros
[trainee@centos7 bin]$ echo $var2
edu is great!
```

Important : Notez que dans le deuxième cas, le reste de la ligne après le mot *fenestros* est mis dans **$var2**.

Code de retour

La commande **read** renvoie un code de retour de dans le cas où elle ne reçoit pas l'information **fin de fichier** matérialisée par les touches `Ctrl+D`.
Le contenu de la variable **var** peut être vide et la valeur du code de retour grâce à l'utilisation de la touche `Entrée` :

```
[trainee@centos7 bin]$ read var
```

`Entrée`

```
[trainee@centos7 bin]$ echo $?

[trainee@centos7 bin]$ echo $var

[trainee@centos7 bin]$
```

Le contenu de la variable **var** peut être vide et la valeur du code de retour **autre que 0** grâce à l'utilisation des touches `Ctrl+D` :

```
[trainee@centos7 bin]$ read var
```

`Ctrl+D`

```
[trainee@centos7 bin]$ echo $?
1
[trainee@centos7 bin]$ echo $var

[trainee@centos7 bin]$
```

La variable IFS

La variable IFS contient par défaut les caractères `Espace`, `Tab` et `Entrée` :

```
[trainee@centos7 bin]$ echo "$IFS" | od -c
0000000      \t  \n  \n
0000004
```

Important : La commande **od** (*Octal Dump*) renvoie le contenu d'un fichier ou de l'entrée standard au format octal. Ceci est utile afin de visualiser les caractères non-imprimables. L'option **-c** permet de sélectionner des caractères ASCII ou des backslash dans le fichier ou dans le contenu fourni à l'entrée standard.

La valeur de cette variable définit donc le séparateur de mots lors de la saisie des contenus des variables avec la commande **read**. La valeur de la variable **IFS** peut être modifiée :

```
[trainee@centos7 bin]$ OLDIFS="$IFS"
[trainee@centos7 bin]$ IFS=":"
[trainee@centos7 bin]$ echo "$IFS" | od -c
0000000   :  \n
0000002
```

De cette façon l'espace redevient un caractère normal :

```
[trainee@centos7 bin]$ read var1 var2 var3
fenestros:edu is:great!
[trainee@centos7 bin]$ echo $var1
fenestros
[trainee@centos7 bin]$ echo $var2
edu is
[trainee@centos7 bin]$ echo $var3
great!
```

Restaurez l'ancienne valeur de IFS avec la commande IFS=“$OLDIFS”

```
[trainee@centos7 bin]$ IFS="$OLDIFS"
[trainee@centos7 bin]$ echo "$IFS" | od -c
0000000      \t  \n  \n
0000004
```

La commande test

La commande **test** peut être utilisée avec deux syntaxes :

test *expression*

ou

[Espace*expression*Espace**]**

Tests de Fichiers

Test	Description
-f fichier	Retourne vrai si fichier est d'un type standard
-d fichier	Retourne vrai si fichier est d'un type répertoire
-r fichier	Retourne vrai si l'utilisateur peut lire fichier
-w fichier	Retourne vrai si l'utilisateur peut modifier fichier
-x fichier	Retourne vrai si l'utilisateur peut exécuter fichier
-e fichier	Retourne vrai si fichier existe
-s fichier	Retourne vrai si fichier n'est pas vide
fichier1 -nt fichier2	Retourne vrai si fichier1 est plus récent que fichier2
fichier1 -ot fichier2	Retourne vrai si fichier1 est plus ancien que fichier2
fichier1 -ef fichier2	Retourne vrai si fichier1 est identique à fichier2

LAB #1

Testez si le fichier **a100** est un fichier ordinaire :

```
[trainee@centos7 bin]$ cd ../training/
[trainee@centos7 training]$ test -f a100
[trainee@centos7 training]$ echo $?

[trainee@centos7 training]$ [ -f a100 ]
[trainee@centos7 training]$ echo $?
```

Testez si le fichier a101 existe :

```
[trainee@centos7 training]$ [ -f a101 ]
```

```
[trainee@centos7 training]$ echo $?
1
```

Testez si /home/trainee/training est un répertoire :

```
[trainee@centos7 training]$ [ -d /home/trainee/training ]
[trainee@centos7 training]$ echo $?
```

Tests de chaînes de caractère

Test	Description
-n chaîne	Retourne vrai si chaîne n'est pas de longueur 0
-z chaîne	Retourne vrai si chaîne est de longueur 0
string1 = string2	Retourne vrai si string1 est égale à string2
string1 != string2	Retourne vrai si string1 est différente de string2
string1	Retourne vrai si string1 n'est pas vide

LAB #2

Testez si les deux chaînes sont égales :

```
[trainee@centos7 training]$ string1="root"
[trainee@centos7 training]$ string2="fenestros"
[trainee@centos7 training]$ [ $string1 = $string2 ]
[trainee@centos7 training]$ echo $?
1
```

Testez si la string1 n'a pas de longueur 0 :

```
[trainee@centos7 training]$ [ -n $string1 ]
[trainee@centos7 training]$ echo $?
```

Testez si la string1 a une longueur de 0 :

```
[trainee@centos7 training]$ [ -z $string1 ]
[trainee@centos7 training]$ echo $?
1
```

Tests sur des nombres

Test	Description
value1 -eq value2	Retourne vrai si value1 est égale à value2
value1 -ne value2	Retourne vrai si value1 n'est pas égale à value2
value1 -lt value2	Retourne vrai si value1 est inférieure à value2
value1 -le value2	Retourne vrai si value1 est inférieur ou égale à value2
value1 -gt value2	Retourne vrai si value1 est supérieure à value2
value1 -ge value2	Retourne vrai si value1 est supérieure ou égale à value2

LAB #3

Comparez les deux nombres **value1** et **value2** :

```
[trainee@centos7 training]$ read value1
35
[trainee@centos7 training]$ read value2
23
[trainee@centos7 training]$ [ $value1 -lt $value2 ]
[trainee@centos7 training]$ echo $?
1
[trainee@centos7 training]$ [ $value2 -lt $value1 ]
[trainee@centos7 training]$ echo $?

[trainee@centos7 training]$ [ $value2 -eq $value1 ]
[trainee@centos7 training]$ echo $?
```

```
1
```

Les opérateurs

Test	Description
!expression	Retourne vrai si expression est fausse
expression1 -a expression2	Représente un et logique entre expression1 et expression2
expression1 -o expression2	Représente un ou logique entre expression1 et expression2
\(expression\)	Les parenthèses permettent de regrouper des expressions

LAB #4

Testez si $file n'est pas un répertoire :

```
[trainee@centos7 training]$ file=a100
[trainee@centos7 training]$ [ ! -d $file ]
[trainee@centos7 training]$ echo $?
```

Testez si $directory est un répertoire **et** si l'utilisateur à le droit de le traverser :

```
[trainee@centos7 training]$ directory=/usr
[trainee@centos7 training]$ [ -d $directory -a -x $directory ]
[trainee@centos7 training]$ echo $?
```

Testez si l'utilisateur peut écrire dans le fichier a100 **et** /usr est un répertoire **ou** /tmp est un répertoire :

```
[trainee@centos7 training]$ [ -w a100 -a \( -d /usr -o -d /tmp \) ]
[trainee@centos7 training]$ echo $?
```

Tests d'environnement utilisateur

Test	Description
-o option	Retourne vrai si l'option du shell “option” est activée

LAB #5

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

La commande [[expression]]

La commande **[[EspaceexpressionEspace]]** est une amélioration de la commande **test**. Les opérateurs de la commande test sont compatibles avec la commande [[expression]] sauf **-a** et **-o** qui sont remplacés par **&&** et **||** respectivement :

Test	Description
!expression	Retourne vrai si expression est fausse
expression1 && expression2	Représente un **et** logique entre expression1 et expression2
expression1 || expression2	Représente un **ou** logique entre expression1 et expression2
(expression)	Les parenthèses permettent de regrouper des expressions

D'autres opérateurs ont été ajoutés :

Test	Description
string = modele	Retourne vrai si chaîne correspond au modèle
string != modele	Retourne vrai si chaîne ne correspond pas au modèle
string1 < string2	Retourne vrai si string1 est lexicographiquement avant string2
string1 > string2	Retourne vrai si string1 est lexicographiquement après string2

LAB #6

Testez si l'utilisateur peut écrire dans le fichier a100 **et** /usr est un répertoire **ou** /tmp est un répertoire :

```
[trainee@centos7 training]$ [[ -w a100 && ( -d /usr || -d /tmp ) ]]
[trainee@centos7 training]$ echo $?
```

Opérateurs du shell

Opérateur	Description
Commande1 && Commande2	Commande 2 est exécutée si la première commande renvoie un code vrai
Commande1 || Commande2	Commande 2 est exécutée si la première commande renvoie un code faux

LAB #7

```
[trainee@centos7 training]$ [[ -d /root ]] && echo "The root directory exists"
The root directory exists
[trainee@centos7 training]$ [[ -d /root ]] || echo "The root directory exists"
[trainee@centos7 training]$
```

L'arithmétique

La commande expr

La commande **expr** prend la forme :

expr Espace value1 Espace *opérateur* Espace value2 Entrée

ou

expr Tab value1 Tab *opérateur* Tab value2 Entrée

ou

expr Espace chaîne Espace : Espace *expression_régulière* Entrée

ou

expr Tab chaîne Tab : Tab *expression_régulière* Entrée

Opérateurs Arithmétiques

Opérateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo
\(\)	Parenthèses

Opérateurs de Comparaison

Opérateur	Description
\<	Inférieur
\<=	Inférieur ou égal
\>	Supérieur
\>=	Supérieur ou égal
=	égal
!=	inégal

Opérateurs Logiques

Opérateur	Description
|	ou logique
\&	et logique

LAB #8

Ajoutez 2 à la valeur de $x :

```
[trainee@centos7 training]$ x=2
[trainee@centos7 training]$ expr $x + 2
4
```

Si les espaces sont retirés, le résultat est tout autre :

```
[trainee@centos7 training]$ expr $x+2
2+2
```

Les opérateurs doivent être protégés :

```
[trainee@centos7 training]$ expr $x * 2
expr: syntax error
[trainee@centos7 training]$ expr $x \* 2
4
```

Mettez le résultat d'un calcul dans une variable :

```
[trainee@centos7 training]$ resultat=`expr $x + 10`
[trainee@centos7 training]$ echo $resultat
12
```

La commande let

La commande let est l'équivalent de la commande ((expression)). La commande ((expression)) est une amélioration de la commande **expr** :

- plus grand nombre d'opérateurs
- pas besoin d'espaces ou de tabulations entre les arguments
- pas besoin de préfixer les variables d'un $
- les caractères spéciaux du shell n'ont pas besoin d'être protégés

- les affectations se font dans la commande
- exécution plus rapide

Opérateurs Arithmétiques

Opérateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo
^	Puissance

Opérateurs de comparaison

Opérateur	Description
<	Inférieur
<=	Inférieur ou égal
>	Supérieur
>=	Supérieur ou égal
==	égal
!=	inégal

Opérateurs Logiques

Opérateur	Description
&&	et logique
||	ou logique
!	négation logique

Opérateurs travaillant sur les bits

Opérateur	Description
~	négation binaire
>>	décalage binaire à droite
<<	décalage binaire à gauche
&	et binaire
|	ou binaire
^	ou exclusif binaire

LAB #9

```
[trainee@centos7 training]$ x=2
[trainee@centos7 training]$ ((x=$x+10))
[trainee@centos7 training]$ echo $x
12
[trainee@centos7 training]$ ((x=$x+20))
[trainee@centos7 training]$ echo $x
32
```

Structures de contrôle

If

La syntaxe de la commande If est la suivante :

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

ou :

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

ou encore :

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

fi
```

LAB #10

Créez le script **user_check** suivant :

```
#!/bin/bash
if [ $# -ne 1 ] ; then
  echo "Mauvais nombre d'arguments"
  echo "Usage : $0 nom_utilisateur"
```

```
  exit 1
fi
if grep "^$1:" /etc/passwd > /dev/null
then
  echo "Utilisateur $1 est défini sur ce système"
else
  echo "Utilisateur $1 n'est pas défini sur ce système"
fi
exit 0
```

Testez-le :

```
[trainee@centos7 training]$ chmod 770 user_check
[trainee@centos7 training]$ ./user_check
Mauvais nombre d'arguments
Usage : ./user_check nom_utilisateur
[trainee@centos7 training]$ ./user_check root
Utilisateur root est défini sur ce système
[trainee@centos7 training]$ ./user_check mickey mouse
Mauvais nombre d'arguments
Usage : ./user_check nom_utilisateur
[trainee@centos7 training]$ ./user_check "mickey mouse"
Utilisateur mickey mouse n'est pas défini sur ce système
```

case

La syntaxe de la commande case est la suivante :

```
case $variable in
modele1) commande
   ...
   ;;
```

```
modele2) commande
   ...
   ;;
modele3 | modele4 | modele5 ) commande
   ...
   ;;
esac
```

Exemple

```
  case "$1" in
      start)
          start
          ;;
      stop)
          stop
          ;;
      restart|reload)
          stop
          start
          ;;
      status)
          status
          ;;
      *)
          echo $"Usage: $0 {start|stop|restart|status}"
          exit 1
esac
```

Important : L'exemple indique que dans le cas où le premier argument qui suit le nom du script contenant la clause **case** est **start**, la fonction *start* sera exécutée. La fonction *start* n'a pas besoin d'être définie dans **case** et est donc en règle générale définie en début de script. La même

logique est appliquée dans le cas où le premier argument est **stop**, **restart** ou **reload** et **status**. Dans tous les autres cas, représentés par une étoile, **case** affichera la ligne **Usage: $0 {start|stop|restart|status}** où $0 est remplacé par le nom du script.

Boucles

for

La syntaxe de la commande for est la suivante :

```
for variable in liste_variables
do
    commande(s)
done
```

while

La syntaxe de la commande while est la suivante :

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

Exemple

```
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
```

Scripts de Démarrage

Quand Bash est appelé en tant que shell de connexion, il exécute des scripts de démarrage dans l'ordre suivant :

- **/etc/profile**,
- **~/.bash_profile** ou **~/.bash_login** ou **~/.profile** selon la distribution,

Dans le cas de RHEL/CentOS, le système exécute le fichier **~/.bash_profile**.

Quand un shell de login se termine, Bash exécute le fichier **~/.bash_logout** si celui-ci existe.

Quand bash est appelé en tant que shell interactif qui n'est pas un shell de connexion, il exécute le script **~/.bashrc**.

LAB #11

A faire : En utilisant vos connaissances acquises dans ce module, expliquez les scripts suivants ligne par ligne.

~/.bash_profile

```
[trainee@centos7 training]$ cat ~/.bash_profile
# .bash_profile
```

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

# User specific environment and startup programs

PATH=$PATH:$HOME/.local/bin:$HOME/bin

export PATH
```

~/.bashrc

```
[trainee@centos7 training]$ cat ~/.bashrc
# .bashrc

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

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

# User specific aliases and functions
```