Dernière mise-à-jour : 2020/01/30 03:27

LUF105 - 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 Ubuntu 16.04 : /bin/sh
csh	C-Shell	1978	Bill Joy	csh	Le shell BSD. Sous Ubuntu 16.04 : /bin/csh
tcsh	Tenex C-Shell	1979	Ken Greer	tcsh	Un dérivé du shell csh. Sous Ubuntu 16.04 : /usr/bin/tcsh
ksh	Korn Shell	1980	David Korn	ksh	Uniquement libre depuis 2005. Sous Ubuntu 16.04 : /usr/bin/ksh
bash	Bourne Again Shell	1987	Brian Fox	bash	Le shell par défaut de Linux et de MacOS X. Sous Ubuntu 16.04 : /bin/bash
zsh	Z Shell	1990	Paul Falstad	zsh	Zsh est plutôt orienté pour l'interactivité avec l'utilisateur. Sous Ubuntu 16.04 : /usr/bin/zsh

Sous Ubuntu 16.04 le shell **/bin/sh** est un lien symbolique vers **/bin/dash** :

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

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

[stextbox id='black' image='null'] **Important** : Notez que dans ce cas l'alias **ls** est en effet un alias qui utilise la **commande** ls elle-même. [/stextbox]

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

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

[stextbox id='black' image='null'] **Important** : 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**. [/stextbox]

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

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

[stextbox id='black' image='null'] **Important** : 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. [/stextbox]

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

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

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

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

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

```
trainee@ubuntu1604:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

```
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-network:x:101:103:systemd Network Management,,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false
syslog:x:104:108::/home/syslog:/bin/false
_apt:x:105:65534::/nonexistent:/bin/false
messagebus:x:106:110::/var/run/dbus:/bin/false
uuidd:x:107:111::/run/uuidd:/bin/false
lightdm:x:108:114:Light Display Manager:/var/lib/lightdm:/bin/false
whoopsie:x:109:116::/nonexistent:/bin/false
avahi-autoipd:x:110:119:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false
avahi:x:111:120:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/bin/false
colord:x:113:123:colord colour management daemon,,,:/var/lib/colord:/bin/false
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/false
hplip:x:115:7:HPLIP system user,,,:/var/run/hplip:/bin/false
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/bin/false
```

```
pulse:x:117:124:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false
saned:x:119:127::/var/lib/saned:/bin/false
usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false
trainee:x:1000:1000:trainee,,,:/home/trainee:/bin/bash
sshd:x:121:65534::/var/run/sshd:/usr/sbin/nologin
```

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

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

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

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

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

[stextbox id='black' image='null'] **Important** : Notez que le mot de passe saisi ne sera **pas** visible. [/stextbox]

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

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

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

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

[stextbox id='black' image='null'] **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. [/stextbox]

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

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

[stextbox id='black' image='null'] **Important** : Notez que le mot de passe saisi ne sera **pas** visible. [/stextbox]

Le Prompt

Le prompt d'un utilisateur dépend de son statut :

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

[stextbox id='black' image='null'] **Important**: L'historique est spécifique à chaque utilisateur. [/stextbox]

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

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

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

Le paramétrage de la fonction du rappel des commandes est fait pour chaque utilisateur individuellement dans le fichier **~/.bashrc** où **~/** indique le répertoire personnel de l'utilisateur concerné. Dans ce fichier, les variables concernant le rappel des commandes peuvent être définis. Le plus important est **HISTSIZE** :

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

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@ubuntu1604:~$ nl .bash_history | more
     1  sudo su -
     2  vi vitext
     3  view vitext
     4  vi vitext
     5  vi .exrc
     6  vi vitext
     7  clear
     8  stty -a
     9  date
    10  locale
    11  LANG=en_GB.UTF-8
    12  export LANG
    13  locale
    14  date
    15  LC_ALL=en_GB.UTF-8
```

```
    16  export LC_ALL
    17  locale
    18  date
    19  who
    20  df
    21  df -h
    22  free
    23  free -h
--More--
```

[stextbox id='black' image='null'] **Important** : Notez l'utilisation de la commande **nl** pour numéroter les lignes de l'affichage du contenu du fichier **.bash_history**. [/stextbox]

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

[stextbox id='black' image='null'] **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. [/stextbox]

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

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

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

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

[stextbox id='black' image='null'] **Important** : Notez que le caractère * remplace un caractère ou une suite de caractères. [/stextbox]

Caractère ?

Créez maintenant les fichiers f52 et f62 :

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

Saisissez ensuite la commande suivante :

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

[stextbox id='black' image='null'] **Important** : Notez que le caractère **?** remplace **un seul** caractère. [/stextbox]

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

Ensuite saisissez les commandes suivantes et notez le résultat :

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

[stextbox id='black' image='null'] **Important** : Notez ici que tous les fichiers commençant par les lettres **a**, **b**, **c**, **d**, **e** ou **f** sont affichés à l'écran. [/stextbox]

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

[stextbox id='black' image='null'] **Important** : Notez ici que tous les fichiers sont affichés à l'écran, à l'exception d'un fichier commençant par la lettre **a** . [/stextbox]

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

[stextbox id='black' image='null'] **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**. [/stextbox]

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

[stextbox id='black' image='null'] **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]**. [/stextbox]

L'option extglob

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

```
trainee@ubuntu1604:~/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@ubuntu1604:~/training$ shopt
autocd          off
cdable_vars     off
cdspell         off
checkhash       off
checkjobs       off
checkwinsize    on
cmdhist         on
compat31        off
compat32        off
compat40        off
compat41        off
compat42        off
complete_fullquote  on
direxpand       off
dirspell        off
dotglob         off
execfail        off
expand_aliases  on
extdebug        off
extglob         on
extquote        on
failglob        off
force_fignore   on
```

```
globstar        off
globasciiranges off
gnu_errfmt      off
histappend      on
histreedit      off
histverify      off
hostcomplete    off
huponexit       off
interactive_comments    on
lastpipe        off
lithist         off
login_shell     on
mailwarn        off
no_empty_cmd_completion off
nocaseglob      off
nocasematch     off
nullglob        off
progcomp        on
promptvars      on
restricted_shell    off
shift_verbose   off
sourcepath      on
xpg_echo        off
```

?(expression)

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

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

Saisissez la commande suivante :

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

[stextbox id='black' image='null'] **Important** : Notez ici que la commande affiche les fichiers ayant un nom contenant 0 ou 1 occurence de la chaîne **123**. [/stextbox]

*(expression)

Saisissez la commande suivante :

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

[stextbox id='black' image='null'] **Important** : Notez ici que la commande affiche les fichiers ayant un nom contenant de 0 jusqu'à x occurences de la chaîne **123**. [/stextbox]

+(expression)

Saisissez la commande suivante :

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

[stextbox id='black' image='null'] **Important** : Notez ici que la commande affiche les fichiers ayant un nom contenant entre 1 et x occurences de la chaîne **123**. [/stextbox]

@(expression)

Saisissez la commande suivante :

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

[stextbox id='black' image='null'] **Important** : Notez ici que la commande affiche les fichiers ayant un nom contenant 1 seule occurence de la chaîne **123**. [/stextbox]

!(expression)

Saisissez la commande suivante :

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

[stextbox id='black' image='null'] **Important** : Notez ici que la commande n'affiche que les fichiers ayant un nom qui ne contient **pas** la chaîne **123**. [/stextbox]

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

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

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

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

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

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

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@ubuntu1604:~/training$ date > file
```

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

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

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

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

[stextbox id='black' image='null'] **Important** : Notez que la sortie standard ne peut être redirigée que dans **une seule direction**. [/stextbox]

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

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

[stextbox id='black' image='null'] **Important** : Il est à noter qu'il est possible de relier plusieurs tubes dans la même commande. [/stextbox]

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

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

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

```
trainee@ubuntu1604:~$ date | tee fichier1 > fichier2
trainee@ubuntu1604:~$ cat fichier1
Mon 28 Nov 16:15:56 CET 2016
trainee@ubuntu1604:~$ cat fichier2
Mon 28 Nov 16:15:56 CET 2016
```

[stextbox id='black' image='null'] **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. [/stextbox]

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@ubuntu1604:~$ echo date
```

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

[stextbox id='black' image='null'] **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. [/stextbox]

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.

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

Par exemple :

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

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

trainee@ubuntu1604:~$ locale
LANG=en_GB.UTF-8
LANGUAGE=
LC_CTYPE="en_GB.UTF-8"
LC_NUMERIC="en_GB.UTF-8"
LC_TIME="en_GB.UTF-8"
LC_COLLATE="en_GB.UTF-8"
LC_MONETARY="en_GB.UTF-8"
LC_MESSAGES="en_GB.UTF-8"
```

```
LC_PAPER="en_GB.UTF-8"
LC_NAME="en_GB.UTF-8"
LC_ADDRESS="en_GB.UTF-8"
LC_TELEPHONE="en_GB.UTF-8"
LC_MEASUREMENT="en_GB.UTF-8"
LC_IDENTIFICATION="en_GB.UTF-8"
LC_ALL=en_GB.UTF-8
```

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
$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@ubuntu1604:~$ env
LC_PAPER=fr_FR.UTF-8
LC_ADDRESS=fr_FR.UTF-8
XDG_SESSION_ID=1
LC_MONETARY=fr_FR.UTF-8
TERM=xterm-256color
SHELL=/bin/bash
SSH_CLIENT=10.0.2.2 40266 22
```

```
LC_NUMERIC=fr_FR.UTF-8
OLDPWD=/home/trainee/training
SSH_TTY=/dev/pts/8
LC_ALL=en_GB.UTF-8
USER=trainee
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37
;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=0
1;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31
:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*
.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=
01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.jpg=01;35:*.jpeg=01;35
:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.ti
ff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=
01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;3
5:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl
=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36
:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.w
av=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
LC_TELEPHONE=fr_FR.UTF-8
MAIL=/var/mail/trainee
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
QT_QPA_PLATFORMTHEME=appmenu-qt5
LC_IDENTIFICATION=fr_FR.UTF-8
PWD=/home/trainee
LANG=en_US.UTF-8
LC_MEASUREMENT=fr_FR.UTF-8
SHLVL=1
HOME=/home/trainee
LOGNAME=trainee
SSH_CONNECTION=10.0.2.2 40266 10.0.2.15 22
LESSOPEN=| /usr/bin/lesspipe %s
XDG_RUNTIME_DIR=/run/user/1000
LESSCLOSE=/usr/bin/lesspipe %s %s
LC_TIME=fr_FR.UTF-8
```

```
LC_NAME=fr_FR.UTF-8
_=/usr/bin/env
```

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@ubuntu1604:~$ set -o
allexport       off
braceexpand     on
emacs           on
errexit         off
errtrace        off
functrace       off
hashall         on
histexpand      on
history         on
ignoreeof       off
interactive-comments    on
keyword         off
monitor         on
noclobber       off
noexec          off
```

```
noglob          off
nolog           off
notify          off
nounset         off
onecmd          off
physical        off
pipefail        off
posix           off
privileged      off
verbose         off
vi              off
xtrace          off
```

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

set -o allexport [Entrée]

Par exemple :

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

[stextbox id='black' image='null'] **Important** : Notez que l'option **noclobber** peut être contournée en utilisant la redirection suivi par le caractère **|**. [/stextbox]

noglob

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

[stextbox id='black' image='null'] **Important** : Notez que l'effet du caractère spécial est annulé sous l'influence de l'option **noglob**. [/stextbox]

nounset

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

trainee@ubuntu1604:~$
```

[stextbox id='black' image='null'] **Important** : 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. [/stextbox]

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@ubuntu1604:~$ script
Script started, file is typescript
trainee@ubuntu1604:~$ pwd
/home/trainee
trainee@ubuntu1604:~$ ls
aac  codes      Downloads         fichier1  file1     Public     typescript  xyz
abc  Desktop    errorlog          fichier2  Music     Templates  Videos
bca  Documents  examples.desktop  file      Pictures  training   vitext
trainee@ubuntu1604:~$ exit
exit
Script done, file is typescript
trainee@ubuntu1604:~$ cat typescript
Script started on Tue 29 Nov 2016 03:57:47 CET
trainee@ubuntu1604:~$ pwd
/home/trainee
trainee@ubuntu1604:~$ ls
aac  codes      Downloads         fichier1  file1     Public     typescript  xyz
abc  Desktop    errorlog          fichier2  Music     Templates  Videos
bca  Documents  examples.desktop  file      Pictures  training   vitext
trainee@ubuntu1604:~$ exit
exit

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

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@ubuntu1604:~$ vi myscript
trainee@ubuntu1604:~$ /bin/bash myscript
/home/trainee
aac  codes  Downloads      fichier1  file1      Pictures   training    vitext
abc  Desktop    errorlog      fichier2  myscript  Public      typescript  xyz
bca  Documents  examples.desktop  file        Music      Templates  Videos
```

Lancez ensuite le script en redirigeant son entrée standard :

```
trainee@ubuntu1604:~$ /bin/bash < myscript
/home/trainee
aac  codes  Downloads      fichier1  file1      Pictures   training    vitext
abc  Desktop    errorlog      fichier2  myscript  Public      typescript  xyz
bca  Documents  examples.desktop  file        Music      Templates  Videos
```

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

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

Dans le cas d'Ubuntu, ceci n'est pas le cas. La raison pour ceci devient évidente en regardant le fichier **.profile** dans **/home/trainee**:

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

```
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
```

Comme on peut constater, la valeur de PATH ne contiendra $HOME/bin que dans le cas où le répertoire existe :

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

Pour modifier la valeur de PATH, créez le répertoire $HOME/bin et rechargez le fichier .profile :

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

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

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

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

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

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

- ./myscript
- . myscript

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

[stextbox id='black' image='null'] **A faire** : Notez bien la différence entre les sorties de cette dernière commande et la précédente. Expliquez pourquoi. [/stextbox]

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

[stextbox id='black' image='null'] **Important**: 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. [/stextbox]

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

[stextbox id='black' image='null'] **Important** : Notez que dans le deuxième cas, le reste de la ligne après le mot *fenestros* est mis dans **$var2**. [/stextbox]

Code de retour

La commande **read** renvoie un code de retour de **0** 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 **0** grâce à l'utilisation de la touche Entrée :

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

↵ Entrée

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

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

Ctrl+D

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

trainee@ubuntu1604:~/bin$
```

La variable IFS

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

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

[stextbox id='black' image='null'] **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. [/stextbox]

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

De cette façon l'espace redevient un caractère normal :

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

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

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

```
trainee@ubuntu1604:~/bin$ IFS="$OLDIFS"
trainee@ubuntu1604:~/bin$ 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

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

Testez si le fichier a101 existe :

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

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

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

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

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

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

Testez si la string1 a une longueur de 0 :

```
trainee@ubuntu1604:~/training$ [ -z $string1 ]
trainee@ubuntu1604:~/training$ 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@ubuntu1604:~/training$ read value1
35
trainee@ubuntu1604:~/training$ read value2
23
trainee@ubuntu1604:~/training$ [ $value1 -lt $value2 ]
trainee@ubuntu1604:~/training$ echo $?
1
trainee@ubuntu1604:~/training$ [ $value2 -lt $value1 ]
trainee@ubuntu1604:~/training$ echo $?
0
trainee@ubuntu1604:~/training$ [ $value2 -eq $value1 ]
trainee@ubuntu1604:~/training$ echo $?
1
```

Les opérateurs

Test	Description
!expression	Retourne vrai si expression est fausse

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

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

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

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

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

Tests d'environnement utilisateur

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

LAB #5

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

La commande [[expression]]

La commande **[[Espace expression Espace]]** 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@ubuntu1604:~/training$ [[ -w a100 && ( -d /usr || -d /tmp ) ]]
trainee@ubuntu1604:~/training$ echo $?
```

```
0
```

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

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

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

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

Les opérateurs doivent être protégés :

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

Mettez le résultat d'un calcul dans une variable :

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

Opérateur	Description
&	et binaire
|	ou binaire
^	ou exclusif binaire

LAB #9

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

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@ubuntu1604:~/training$ chmod 770 user_check
trainee@ubuntu1604:~/training$ ./user_check
Mauvais nombre d'arguments
Usage : ./user_check nom_utilisateur
trainee@ubuntu1604:~/training$ ./user_check root
Utilisateur root est défini sur ce système
trainee@ubuntu1604:~/training$ ./user_check mickey mouse
Mauvais nombre d'arguments
Usage : ./user_check nom_utilisateur
trainee@ubuntu1604:~/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
```

[stextbox id='black' image='null'] **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. [/stextbox]

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 d'Ubuntu, le système exécute le fichier **~/.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

[stextbox id='black' image='null'] **A faire** : En utilisant vos connaissances acquises dans ce module, expliquez les scripts suivants ligne par ligne. [/stextbox]

~/.profile

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

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

```
#umask 022

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

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

~/.bashrc

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

# If not running interactively, don't do anything
case $- in
    *i*) ;;
      *) return;;
esac

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

# append to the history file, don't overwrite it
```

```
shopt -s histappend

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

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

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

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

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

# set a fancy prompt (non-color, unless we know we "want" color)
case "$TERM" in
    xterm-color|*-256color) color_prompt=yes;;
esac

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

if [ -n "$force_color_prompt" ]; then
    if [ -x /usr/bin/tput ] && tput setaf 1 >&/dev/null; then
```

```
    # We have color support; assume it's compliant with Ecma-48
    # (ISO/IEC-6429). (Lack of such support is extremely rare, and such
    # a case would tend to support setf rather than setaf.)
    color_prompt=yes
    else
    color_prompt=
    fi
fi

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

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

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

    alias grep='grep --color=auto'
    alias fgrep='fgrep --color=auto'
```

```
    alias egrep='egrep --color=auto'
fi

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

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

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

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

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

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

```
fi
```

<html>

Copyright © 2004-2018 Hugh Norris.

</html>