Niveau : Utilisateur	Numéro de la Leçon	Dernière Modification
1/4	<progrecss 5/5 style=inline />	2020/01/30 03:28

Le 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 Linux : /bin/sh
csh	C-Shell	1978	Bill Joy	csh	Le shell BSD. Sous Linux : /bin/csh
tcsh	Tenex C-Shell	1979	Ken Greer	tcsh	Un dérivé du shell csh. Sous Linux : /bin/tcsh
ksh	Korn Shell	1980	David Korn	ksh	Uniquement libre depuis 2005. Sous Linux : /bin/ksh
bash	Bourne Again Shell	1987	Brian Fox	bash	Le shell par défaut de Linux et de MacOS X. Sous Linux : /bin/bash

Cette unité concerne l'utilisation du shell **bash** sous Linux. Cependant, il peut aussi être utile aux utilisateurs de **ksh** sous UNIX car les commandes sont pratiquement identiques.

Le shell **/bin/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** :

```
root@debian:~# type cd
cd est une primitive du shell
```

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

```
root@debian:~# type ifconfig
ifconfig est /sbin/ifconfig
```

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 :

```
root@debian:~# type ls
ls est /bin/ls
root@debian:~# exit
logout
trainee@debian:~$ type ls
```

```
ls est un alias vers « ls --color=auto »
```

<note important> Notez que dans ce cas l'alias **ls** est en effet un alias qui utilise la **commande** ls elle-même. </note>

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

```
trainee@debian:~$ alias dir='ls -l'
trainee@debian:~$ dir
total 32
-rw-r--r--. 1 trainee trainee    0  8 nov.  15:06 aac
-rw-r--r--. 1 trainee trainee    0  8 nov.  15:06 abc
-rw-r--r--. 1 trainee trainee    0  8 nov.  15:06 bca
drwxr-xr-x. 2 trainee trainee 4096 28 juil. 09:32 Bureau
drwxr-xr-x. 2 trainee trainee 4096 24 avril  2011 Documents
drwxr-xr-x. 2 trainee trainee 4096 24 avril  2011 Images
drwxr-xr-x. 2 trainee trainee 4096 24 avril  2011 Modèles
drwxr-xr-x. 2 trainee trainee 4096 24 avril  2011 Musique
drwxr-xr-x. 2 trainee trainee 4096 24 avril  2011 Public
drwxr-xr-x. 2 trainee trainee 4096 14 nov.  16:06 Téléchargements
drwxr-xr-x. 2 trainee trainee 4096 24 avril  2011 Vidéos
-rw-r--r--. 1 trainee trainee    0  8 nov.  15:06 xyz
```

<note 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**. </note>

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

```
trainee@debian:~$ alias
alias dir='ls -l'
alias ls='ls --color=auto'
```

<note important> Notez que cette liste contient, sans distinction, l'alias **ls** défini 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. </note>

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

```
trainee@debian:~$ \dir
aac  bca     Documents  Modèles  Public          Vidéos
abc  Bureau  Images Musique  Téléchargements  xyz
```

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

```
trainee@debian:~$ unalias dir
trainee@debian:~$ dir
aac  bca     Documents  Modèles  Public          Vidéos
abc  Bureau  Images Musique  Téléchargements  xyz
```

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

```
trainee@debian:~$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
```

```
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
messagebus:x:101:103::/var/run/dbus:/bin/false
Debian-exim:x:102:104::/var/spool/exim4:/bin/false
statd:x:103:65534::/var/lib/nfs:/bin/false
avahi:x:104:107:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
usbmux:x:105:46:usbmux daemon,,,:/home/usbmux:/bin/false
Debian-gdm:x:106:114:Gnome Display Manager:/var/lib/gdm3:/bin/false
saned:x:107:116::/home/saned:/bin/false
hplip:x:108:7:HPLIP system user,,,:/var/run/hplip:/bin/false
trainee:x:1000:1000:trainee,,,:/home/trainee:/bin/bash
vboxadd:x:999:1::/var/run/vboxadd:/bin/false
```

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@debian:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/csh
/bin/sh
/usr/bin/es
/usr/bin/ksh
/bin/ksh
/usr/bin/rc
/usr/bin/tcsh
/bin/tcsh
/usr/bin/esh
/bin/dash
/bin/bash
/bin/rbash
```

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

```
trainee@debian:~$ 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@debian:~$ chsh
Mot de passe :
Changement d'interpréteur de commandes initial pour trainee
Entrez la nouvelle valeur ou « Entrée » pour conserver la valeur proposée
    Interpréteur de commandes initial [/bin/bash]: /bin/sh
```

<note important> Notez que le mot de passe saisi ne sera **pas** visible. </note>

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

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

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

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

<note 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. </note>

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

```
trainee@debian:~$ chsh
Mot de passe :
Changement d'interpréteur de commandes initial pour trainee
Entrez la nouvelle valeur ou « Entrée » pour conserver la valeur proposée
    Interpréteur de commandes initial [/bin/sh]: /bin/bash
```

<note important> Notez que le mot de passe saisi ne sera **pas** visible. </note>

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@debian:~$ history
  ...
  168  ls
  169  history
  170  cat /etc/profile
  171  ls
  172  ls -la
  173  cat .bashrc
  174  cat .profile
  175  cat /etc/bash.bashrc
  176  echo $HISTSIZE
  177  man history
  178  echo $HISTSIZE
  179  echo $HISTFILESIZE
  180  ls -la
  181  vi .bashrc
  182  man bash
  183  history
```

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

```
trainee@debian:~$ ls
aac  bca     Documents  Modèles  Public           Vidéos
abc  Bureau  Images     Musique  Téléchargements  xyz
trainee@debian:~$ !!
ls
aac  bca     Documents  Modèles  Public           Vidéos
abc  Bureau  Images     Musique  Téléchargements  xyz
```

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

```
trainee@debian:~$ !172
ls -la
total 216
drwxr-xr-x. 34 trainee trainee  4096  8 déc.  11:26 .
drwxr-xr-x.  3 root    root     4096 24 avril  2011 ..
-rw-r--r--.  1 trainee trainee     0  8 nov.  15:06 aac
-rw-r--r--.  1 trainee trainee     0  8 nov.  15:06 abc
drwx------.  3 trainee trainee  4096 28 juil. 09:18 .adobe
-rw-------.  1 trainee trainee  1923 28 nov.  15:14 .bash_history
-rw-r--r--.  1 trainee trainee   220 24 avril  2011 .bash_logout
-rw-r--r--.  1 trainee trainee  3206  8 déc.  11:24 .bashrc
-rw-r--r--.  1 trainee trainee     0  8 nov.  15:06 bca
drwxr-xr-x.  2 trainee trainee  4096 28 juil. 09:32 Bureau
drwx------.  3 trainee trainee  4096 28 juil. 09:29 .cache
drwxr-xr-x.  8 trainee trainee  4096 28 juil. 09:30 .config
drwx------.  3 trainee trainee  4096 24 avril  2011 .dbus
-rw-r--r--.  1 trainee trainee    48  8 déc.  10:41 .dmrc
drwxr-xr-x.  2 trainee trainee  4096 24 avril  2011 Documents
-rw-------.  1 trainee trainee    16 28 juil. 09:15 .esd_auth
drwx------.  3 trainee trainee  4096 28 juil. 09:27 .evolution
drwxr-xr-x.  2 trainee trainee  4096  3 oct.  16:30 .fontconfig
drwx------.  5 trainee trainee  4096  8 déc.  10:41 .gconf
drwx------.  2 trainee trainee  4096  8 déc.  11:05 .gconfd
-rw-r-----.  1 trainee trainee     0  5 oct.  15:51 .gksu.lock
```

```
drwx------.  9 trainee trainee  4096 18 oct.  19:48 .gnome2
drwx------.  2 trainee trainee  4096 28 juil. 09:14 .gnome2_private
drwx------.  2 trainee trainee  4096 24 avril  2011 .gnupg
drwxr-xr-x.  2 trainee trainee  4096  4 oct.  18:55 .gstreamer-0.10
-rw-r--r--.  1 trainee trainee   168  8 déc.  10:41 .gtk-bookmarks
drwx------.  2 trainee trainee  4096 24 avril  2011 .gvfs
-rw-------.  1 trainee trainee  6360  8 déc.  10:41 .ICEauthority
drwxr-xr-x.  2 trainee trainee  4096 24 avril  2011 Images
drwxr-xr-x.  3 trainee trainee  4096 28 juil. 09:18 .java
-rw-------.  1 trainee trainee    41  8 déc.  11:16 .lesshst
drwx------.  3 trainee trainee  4096 27 avril  2011 .local
drwx------.  3 trainee trainee  4096 28 juil. 09:20 .macromedia
drwx------.  3 trainee trainee  4096 28 juil. 09:27 .mission-control
drwxr-xr-x.  2 trainee trainee  4096 24 avril  2011 Modèles
drwx------.  4 trainee trainee  4096 28 juil. 09:14 .mozilla
drwxr-xr-x.  2 trainee trainee  4096 24 avril  2011 Musique
drwxr-xr-x.  2 trainee trainee  4096 24 avril  2011 .nautilus
drwx------.  3 trainee trainee  4096 28 juil. 09:29 .pki
-rw-r--r--.  1 trainee trainee   675 24 avril  2011 .profile
drwxr-xr-x.  2 trainee trainee  4096 24 avril  2011 Public
-rw-------.  1 trainee trainee   218  2 déc.  10:21 .recently-used.xbel
drwxr-xr-x.  2 trainee trainee  4096  3 oct.  13:44 .shcache
drwxr-xr-x.  2 trainee trainee  4096 14 nov.  16:06 Téléchargements
drwx------.  3 trainee trainee  4096 28 juil. 09:29 .thumbnails
drwx------.  2 trainee trainee  4096 24 avril  2011 .update-notifier
drwxr-xr-x.  2 root    root     4096  3 oct.  16:29 .ure
-rw-r--r--.  1 trainee trainee     5  8 déc.  10:41 .vboxclient-display.pid
-rw-r--r--.  1 trainee trainee     5  8 déc.  10:41 .vboxclient-seamless.pid
drwxr-xr-x.  2 trainee trainee  4096 24 avril  2011 Vidéos
-rw-------.  1 trainee trainee 20631  8 déc.  11:26 .xsession-errors
-rw-------.  1 trainee trainee  1324  2 déc.  13:47 .xsession-errors.old
-rw-r--r--.  1 trainee trainee     0  8 nov.  15:06 xyz
```

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

Sous Debian, le paramétrage de la fonction du rappel des commandes est défini par des variables système. Par exemple le nombre de commandes mémorisées est défini par la variable $HISTSIZE :

```
trainee@debian:~$ echo $HISTSIZE
500
```

Comme vous pouvez le constater, la valeur par défaut est de **500**.

Pour augmenter le nombre de commandes mémorisées, l'utilisateur peut éditer son fichier **~/.bashrc**, ou **~/** indique le répertoire personnel de l'utilisateur concerné, en ajoutant la ligne suivante à l'emplacement prévu à cet effet :

```
# don't put duplicate lines in the history. See bash(1) for more options
# don't overwrite GNU Midnight Commander's setting of `ignorespace'.
HISTCONTROL=$HISTCONTROL${HISTCONTROL+:}ignoredups
# ... or force ignoredups and ignorespace
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)

export HISTSIZE=1000 #<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< La ligne ajoutée

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

```
...
```

Afin que le fichier **~/.bashrc** soit relu, il convient d'utiliser la commande **source** :

```
trainee@debian:~$ source ~/.bashrc
```

Le résultat de cette opération est l'augmentation du nombre de commandes mémorisées :

```
trainee@debian:~$ echo $HISTSIZE
1000
```

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

Les commandes mémorisées sont stockées dans le fichier **~/.bash_history** :

```
trainee@debian:~$ nl .bash_history
   ...
   150  touch aac abc bca xyz
   151  ls
   152  /etc/init.d/networking restart
   153  su -
   154  ls
   155  more /etc/group
   156  more --help
   157  find acc
   158  find aac
   159  find --help
   160  mount --help
   161  umount --help
   162  view Téléchargements/vitexte
   163  vi Téléchargements/vitexte
   164  view Téléchargements/vitexte
   165  vi Téléchargements/vitexte
```

```
   166  view Téléchargements/vitexte
   167  vi Téléchargements/vitexte
```

<note important> Notez l'utilisation de la commande **nl** pour numéroter les lignes de l'affichage du contenu du fichier **.bash_history**. </note>

La comparaison du contenu de ce fichier avec la sortie de la commande **history** démontre que les deux sont diffèrents. En effet, le fichier **.bash_history** ne contient pas les lignes **168** à **183** de la sortie de la commande **history**.

<note important> Les lignes **168** et supérieures ne seront inscrites dans le fichier .bash_history qu'au moment de la fermeture du terminal dans lequel les commandes ont étaient saisies. </note>

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 :

```
$ more .b[Tab][Tab][Tab]
```

```
trainee@debian:~$ more .bash
.bash_history  .bash_logout   .bashrc
```

<note important> Notez qu'en appuyant sur la touche Tab trois fois le shell propose 3 possibilités de complétion de nom de fichier. En effet, sans plus d'information, le shell ne sait pas quel fichier doit être ouvert. </note>

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@debian:~$ mo
mogrify      montage      mount        mousetweaks
```

```
mono         more         mountpoint
```

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 **formation**. Ensuite créez dans ce répertoire 5 fichiers nommés respectivement f1, f2, f3, f4 et f5 :

```
trainee@debian:~$ mkdir formation
trainee@debian:~$ cd formation
trainee@debian:~/formation$ touch f1 f2 f3 f4 f5
trainee@debian:~/formation$ ls -l
total 0
-rw-r--r--. 1 trainee trainee 0  8 déc.  12:15 f1
-rw-r--r--. 1 trainee trainee 0  8 déc.  12:15 f2
-rw-r--r--. 1 trainee trainee 0  8 déc.  12:15 f3
-rw-r--r--. 1 trainee trainee 0  8 déc.  12:15 f4
```

```
-rw-r--r--. 1 trainee trainee 0  8 déc.  12:15 f5
```

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

```
trainee@debian:~/formation$ echo f*
f1 f2 f3 f4 f5
```

<note important> Notez que le caractère * remplace un caractère ou une suite de caractères. </note>

Caractère ?

Créez maintenant les fichiers f52 et f62 :

```
trainee@debian:~/formation$ touch f52 f62
```

Saisissez ensuite la commande suivante :

```
trainee@debian:~/formation$ echo f?2
f52 f62
```

<note important> Notez que le caractère **?** remplace **un seul** caractère. </note>

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

Joker	Description
[!m-t]	Représente un caractère autre que m ou n .... t

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

```
trainee@debian:~/formation$ touch a100
```

Ensuite saisissez les commandes suivantes et notez le résultat :

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

<note important> Notez ici que tous les fichiers commençant par les lettres **a**, **b**, **c**, **d**, **e** ou **f** sont affichés à l'écran. </note>

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

<note important> Notez ici que tous les fichiers sont affichés à l'écran, à l'exception d'un fichier commençant par la lettre **a** . </note>

```
trainee@debian:~/formation$ echo [a-b]*
a100
```

<note 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**. </note>

```
trainee@debian:~/formation$ echo [a-f]
[a-f]
```

<note important> Notez que dans ce cas, il n'existe pas de fichiers dénommés **a**, **b**, **c**, **d**, **e** ou **f**. Pour cette raison, n'ayons 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]**. </note>

L'option extglob

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

```
trainee@debian:~/formation$ 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@debian:~/formation$ shopt
autocd          off
cdable_vars     off
cdspell         off
checkhash       off
checkjobs       off
checkwinsize    on
cmdhist         on
compat31        off
compat32        off
compat40        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
lithist         off
login_shell     off
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@debian:~/formation$ touch f f.txt f123.txt f123123.txt f123123123.txt
```

Saisissez la commande suivante :

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

<note important> Notez ici que la commande affiche les fichiers ayant un nom contenant 0 ou 1 occurence de la chaîne **123**. </note>

***(expression)**

Saisissez la commande suivante :

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

<note important> Notez ici que la commande affiche les fichiers ayant un nom contenant de 0 jusqu'à x occurences de la chaîne **123**. </note>

+(expression)

Saisissez la commande suivante :

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

<note important> Notez ici que la commande affiche les fichiers ayant un nom contenant entre 1 et x occurences de la chaîne **123**. </note>

@(expression)

Saisissez la commande suivante :

```
trainee@debian:~/formation$ ls f@(123).txt
f123.txt
```

<note important> Notez ici que la commande affiche les fichiers ayant un nom contenant 1 seule occurence de la chaîne **123**. </note>

!(expression)

Saisissez la commande suivante :

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

<note important> Notez ici que la commande n'affiche que les fichiers ayant un nom qui ne contient **pas** la chaîne **123**. </note>

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 **formation**, vous obtiendrez une fenêtre similaire à celle-ci :

```
trainee@debian:~/formation$ echo * est un caractère spécial
a100 f1 f2 f3 f4 f5 f52 f62 est un caractère spécial
```

Vous noterez que le caractère spécial * a bien été interprété par le shell.

Afin de protéger le caractère *, nous devons utiliser un caractère d'échappement. Commençons par l'utilisation du caractère **** :

```
trainee@debian:~/formation$ echo \* est un caractère spécial
* est un caractère spécial
```

Vous noterez que le caractère spécial * n'a pas été interprété par le shell.

Le même résultat peut être obtenu en utilisant ainsi :

```
trainee@debian:~/formation$ echo "* est un caractère spécial"
* est un caractère spécial
trainee@debian:~/formation$ 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@debian:~/formation$ cd ..
trainee@debian:~$ mkdir codes
trainee@debian:~$ echo $?
0
trainee@debian:~$ touch codes/retour
trainee@debian:~$ rmdir codes
rmdir: échec de suppression de « codes »: Le dossier n'est pas vide
trainee@debian:~$ echo $?
1
```

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

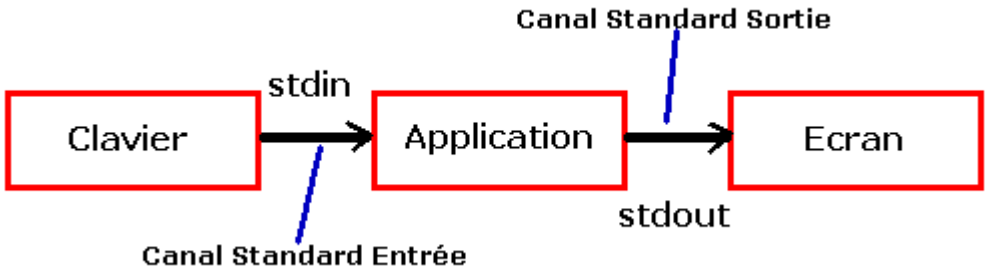
La suppression du répertoire a rencontré une erreur car **codes** contenait le fichier **retour**. Le code retour stocké dans la variable $? est un **un**.

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

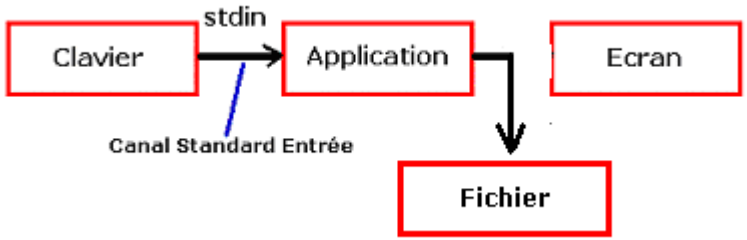
Votre dialogue avec le système Linux utilise des canaux d'entrée et de sortie. On appelle le clavier, le **canal d'entrée standard** et l'écran, le **canal de sortie standard** :



;#; ;#;

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** :

```
$ free > fichier [Entrée]
```

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.

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

```
$ date >> fichier [Entrée]
```

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

```
trainee@debian:~$ cd formation
trainee@debian:~/formation$ free > fichier
trainee@debian:~/formation$ date >> fichier
trainee@debian:~/formation$ cat fichier
             total       used       free     shared    buffers     cached
Mem:       1034472     729100     305372          0      69752     351680
-/+ buffers/cache:     307668     726804
Swap:      1951856          0    1951856
jeudi 8 décembre 2011, 12:27:53 (UTC+0100)
```

<note important> Notez que la sortie standard ne peut être redirigée que dans **une seule direction**. </note>

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 **erreurlog** qui contient les messages d'erreur de l'exécution de la commande **rmdir** :

```
$ rmdir formation/ 2> erreurlog [Entrée]
```

En vous placant dans votre répertoire personnel, et en saisissant cette commande, vous obtiendrez une fenêtre similaire à celle-ci :

```
trainee@debian:~/formation$ cd ..
trainee@debian:~$ rmdir formation/ 2>erreurlog
trainee@debian:~$ cat erreurlog
rmdir: échec de suppression de « formation/ »: Le dossier n'est pas vide
```

En effet l'erreur est générée parce que le répertoire **formation** 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 :

```
$ free > fichier 2>&1 [Entrée]
```

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

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 erreurlog et l'affiche à l'écran :

```
trainee@debian:~$ wc -w < erreurlog
11
```

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@debian:~$ ls | wc -w
```

```
15
```

<note important> Il est à noter qu'il est possible de relier plusieurs tubes dans la même commande. </note>

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@debian:~$ date | tee fichier1
jeudi 8 décembre 2011, 12:30:24 (UTC+0100)
trainee@debian:~$ cat fichier1
jeudi 8 décembre 2011, 12:30:24 (UTC+0100)
```

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

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

```
trainee@debian:~$ date | tee fichier1 > fichier2
trainee@debian:~$ cat fichier1
jeudi 8 décembre 2011, 12:30:43 (UTC+0100)
trainee@debian:~$ cat fichier2
jeudi 8 décembre 2011, 12:30:43 (UTC+0100)
```

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@debian:~$ echo date
date
trainee@debian:~$ echo $(date)
jeudi 8 décembre 2011, 12:32:47 (UTC+0100)
trainee@debian:~$ echo `date`
```

```
jeudi 8 décembre 2011, 12:33:00 (UTC+0100)
```

<note 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. </note>

Chainage de Commandes

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

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

Cet exemple envoie le résultat des trois commandes vers le fichier liste 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.

Variable	Description
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 de l'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 **i10n** 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 française les valeurs de langue-PAYS sont :

- fr-BE = la Belgique francophone,
- fr-CA = le Québec,
- fr-FR = la France,
- fr-LU = le Luxembourg,
- fr-MC = Monaco,
- fr-CH = la Suisse francophone.

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@debian:~$ echo $LC_ALL

trainee@debian:~$ echo $LC_CTYPE

trainee@debian:~$ echo $LANG
fr_FR.utf8
```

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

Variable	Description
$@	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 :

```
root@debian:~# env
SHELL=/bin/bash
TERM=xterm
XDG_SESSION_COOKIE=9dbc42206eca490459754e5100000008-1394376534.132688-1876788925
USER=root
MAIL=/var/mail/root
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
PWD=/root
LANG=fr_FR.UTF-8
SHLVL=1
HOME=/root
LOGNAME=root
DISPLAY=:0.0
XAUTHORITY=/var/run/gdm3/auth-for-trainee-Bb3WgH/database
COLORTERM=gnome-terminal
_=/usr/bin/env
```

La commande peut aussi être utilisée pour fixer une variable lors de l'exécution d'une commande. Lancez **xterm** avec la variable EDITOR fixée à **vim** :

```
root@debian:~# env EDITOR=vim xterm
```

Options de la commande

Les options de cette commande sont :

```
root@debian:~# env --help
Utilisation : env [OPTION]... [-] [NAME=VALUE]... [COMMAND [ARG]...]
Initialise chaque NAME à VALUE dans l'environnement et exécute COMMAND.

  -i, --ignore-environment   démarre avec un environnement vide
  -0, --null                 termine chaque ligne de sortie avec l'octet 0 au
                             lieu d'une ligne nouvelle
  -u, --unset=NAME           enlève la variable de l'environment
      --help     affiche l'aide et quitte
      --version  affiche des informations de version et quitte

Un simple - implique -i. Si aucune COMMAND n'est fournie, affiche
l'environnement résultant.

Signalez les anomalies de « env » à <bug-coreutils@gnu.org>
Page d'accueil de « GNU coreutils » : <http://www.gnu.org/software/coreutils/>
Aide générale sur les logiciels GNU : <http://www.gnu.org/gethelp/>
Traduction de « env » à <http://translationproject.org/team/fr.html>
Pour une documentation complète, lancer « info coreutils 'env invocation' »
```

Options du Shell Bash

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

```
$ set -o [Entrée]
```

Par exemple :

```
trainee@debian:~$ 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@debian:~$ set -o allexport
trainee@debian:~$ set -o
allexport       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@debian:~$ set +o allexport
trainee@debian:~$ set -o
allexport       off
...
```

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@debian:~$ set -o noclobber
trainee@debian:~$ pwd > file
trainee@debian:~$ pwd > file
```

```
bash: file : impossible d'écraser le fichier existant
trainee@debian:~$ pwd >|file
trainee@debian:~$ set +o noclobber
```

<note important> Notez que l'option **noclobber** peut être contournée en utilisant la redirection suivi par le caractère **|**. </note>

noglob

```
trainee@debian:~$ set -o noglob
trainee@debian:~$ echo *
*
trainee@debian:~$ set +o noglob
trainee@debian:~$ echo *
aac abc bca Bureau codes Documents erreurlog fichier1 fichier2 file formation Images Modèles Musique Public
Téléchargements Vidéos xyz
```

<note important> Notez que l'effet du caractère spécial est annulé sous l'influence de l'option **noglob**. </note>

nounset

```
trainee@debian:~$ set -o nounset
trainee@debian:~$ echo $FENESTROS
bash: FENESTROS : variable sans liaison
trainee@debian:~$ set +o nounset
trainee@debian:~$ echo $FENESTROS

trainee@debian:~$
```

<note 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. </note>

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 monscript

soit en redirigeant son entrée standard :

/bin/bash < monscript

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 :

monscript

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 :

. monscript et **./monscript**

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@debian:~$ cd formation/
trainee@debian:~/formation$ script
Script started, file is typescript
trainee@debian:~/formation$ pwd
/home/trainee/formation
trainee@debian:~/formation$ ls
a100  f1  f2  f3  f4  f5  f52  f62  fichier  typescript
trainee@debian:~/formation$ exit
exit
Script done, file is typescript
trainee@debian:~/formation$ cat typescript
```

Le contenu de votre fichier **typescript** sera similaire à cet exemple :

```
Script started on jeu. 08 déc. 2011 12:48:10 CET
trainee@debian:~/formation$ pwd
/home/trainee/formation
trainee@debian:~/formation$ ls
a100  f1  f2  f3  f4  f5  f52  f62  fichier  typescript
trainee@debian:~/formation$ exit
exit

Script done on jeu. 08 déc. 2011 12:48:18 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 **monscript** avec **vi** :

```
$ vi monscript [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@debian:~/formation$ vi monscript
trainee@debian:~/formation$ /bin/bash monscript
/home/trainee/formation
a100  f1  f2  f3  f4  f5  f52  f62  fichier  monscript  typescript
```

Lancez ensuite le script en redirigeant son entrée standard :

```
trainee@debian:~/formation$ /bin/bash < monscript
/home/trainee/formation
a100  f1  f2  f3  f4  f5  f52  f62  fichier  monscript  typescript
```

Pour lancer le script en l'appelant simplement par son nom, son chemin doit être inclus dans votre PATH. Consultez donc le fichier ~/.profile :

```
trainee@debian:~/formation$ cat ../.profile | grep PATH
# set PATH so it includes user's private bin if it exists
    PATH="$HOME/bin:$PATH"
trainee@debian:~/formation$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

Vous constaterez que le répertoire **$HOME/bin**, autrement dit **/home/trainee/bin**, n'a pas été rajouté à votre PATH car ce répertoire n'existe pas. Créez donc le répertoire :

```
trainee@debian:~/formation$ mkdir ~/bin
```

Re-chargez maintenant le fichier **~/.profile** :

```
trainee@debian:~/formation$ source ~/.profile 2>/dev/null
```

et vérifiez votre PATH :

```
trainee@debian:~/formation$ echo $PATH
/home/trainee/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

<note important> Notez la présence du répertoire **/home/trainee/bin** dans votre PATH. </note>

Afin de rendre la modification de la variable **$PATH** permanente entre sessions, éditez le fichier **.profile** ainsi :

```
# ~/.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"
export PATH  # <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< Ajoutez cette ligne
fi
```

Déplacez maintenant votre script dans ce répertoire, rendez-le exécutable pour votre utilisateur et vérifiez qu'il est bien exécutable:

```
trainee@debian:~/formation$ mv monscript ~/bin
trainee@debian:~/formation$ cd ~/bin
trainee@debian:~/bin$ chmod u+x monscript
trainee@debian:~/bin$ ls -l
total 4
-rwxr--r--. 1 trainee trainee 7  8 déc.  12:50 monscript
```

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

```
trainee@debian:~/bin$ cd /tmp
trainee@debian:/tmp$ monscript
/tmp
keyring-yLNomI    orbit-trainee  seahorse-EyQIJZ  virtual-trainee.rihpCF
orbit-Debian-gdm  plugtmp    ssh-waPfps1811
```

Dans les trois cas précédents, le script a été exécuté dans un shell fils. Pour l'exécuter dans le shell en cours, placez-vous dans le répertoire contenant le script et saisissez la commande suivante :

```
$ . monscript [Entrée]
```

Vous devez obtenir un résultat similaire à celui-ci :

```
trainee@debian:/tmp$ cd ~/bin
trainee@debian:~/bin$ . monscript
/home/trainee/bin
monscript
```

Exécutez maintenant le script dans un shell fils :

```
trainee@debian:~/bin$ ./monscript
/home/trainee/bin
monscript
```

<note> Notez bien la différence entre les sorties de cette dernière commande et la précédente. Expliquez pourquoi. </note>

La commande read

<note> Vous êtes actuellement connecté en tant que l'utilisateur **trainee**. Devenez maintenant **root** grâce à la commande **su -** et le mot de passe **fenestros**. </note>

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 :

```
root@debian:~# read var1 var2 var3 var4
fenestros edu est super!
root@debian:~# echo $var1
fenestros
root@debian:~# echo $var2
edu
root@debian:~# echo $var3
est
root@debian:~# echo $var4
super!
```

```
root@debian:~# read var1 var2
fenestros edu est super!
root@debian:~# echo $var1
fenestros
root@debian:~# echo $var2
```

```
edu est super!
```

<note important> Notez que dans le deuxième cas, le reste de la ligne après le mot *fenestros* est mis dans **$var2**. </note>

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 :

```
root@debian:~# read var
fenestros
root@debian:~# echo $?
0
root@debian:~# echo $var
fenestros
```

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 :

```
root@debian:~# read $var
```

Entrée

```
root@debian:~#  echo $?
0
root@debian:~#  echo $var

root@debian:~#
```

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 :

```
root@debian:~#  read var
```

Ctrl+D

```
root@debian:~#  echo $?
1
root@debian:~#  echo $var

root@debian:~#
```

La variable IFS

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

```
root@debian:~# echo "$IFS" | od -c
0000000      \t  \n  \n
0000004
```

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 :

```
root@debian:~# OLDIFS="$IFS"
root@debian:~# echo "$OLDIFS" | od -c
0000000      \t  \n  \n
0000004
root@debian:~# IFS=":"
root@debian:~# echo "$IFS" | od -c
0000000   :  \n
0000002
```

De cette façon l'espace redevient un caractère normal :

```
root@debian:~# read var1 var2 var3
fenestros:edu est:super!
root@debian:~# echo $var1
fenestros
```

```
root@debian:~# echo $var2
edu est
root@debian:~# echo $var3
super!
```

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

```
root@debian:~# IFS="$OLDIFS"
root@debian:~# echo "$IFS" | od -c
0000000      \t  \n  \n
0000004
```

<note 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. </note>

La commande test

<note> Vous êtes actuellement connecté en tant que l'utilisateur **root**. Devenez maintenant **trainee** grâce à la commande **exit**. </note>

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

Test	Description
-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

Exemples

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

```
trainee@debian:~/bin$ cd ../formation
trainee@debian:~/formation$ test -f a100
trainee@debian:~/formation$ echo $?
0
trainee@debian:~/formation$ [ -f a100 ]
trainee@debian:~/formation$ echo $?
0
```

Testez si le fichier a101 existe :

```
trainee@debian:~/formation$ [ -f a101 ]
trainee@debian:~/formation$ echo $?
1
```

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

```
trainee@debian:~/formation$ [ -d /home/trainee/formation ]
```

```
trainee@debian:~/formation$ 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
chaîne1 = chaîne2	Retourne vrai si chaîne1 est égale à chaîne2
chaîne1 != chaîne2	Retourne vrai si chaîne1 est différente de chaîne2
chaîne1	Retourne vrai si chaîne1 n'est pas vide

Exemples

Testez si les deux chaînes sont égales :

```
trainee@debian:~/formation$ chaine1="root"
trainee@debian:~/formation$ chaine2="fenestros"
trainee@debian:~/formation$ [ "chaine1" = "chaine2" ]
trainee@debian:~/formation$ echo $?
1
```

Testez si la chaîne1 n'a pas de longueur 0 :

```
trainee@debian:~/formation$ [ -n "chaine1" ]
trainee@debian:~/formation$ echo $?
0
```

Testez si la chaîne1 a une longueur de 0 :

```
trainee@debian:~/formation$ [ -z "chaine1" ]
trainee@debian:~/formation$ echo $?
```

```
1
```

Tests sur des nombres

Test	Description
valeur1 -eq valeur2	Retourne vrai si valeur1 est égale à valeur2
valeur1 -ne valeur2	Retourne vrai si valeur1 n'est pas égale à valeur2
valeur1 -lt valeur2	Retourne vrai si valeur1 est inférieure à valeur2
valeur1 -le valeur2	Retourne vrai si valeur1 est inférieur ou égale à valeur2
valeur1 -gt valeur2	Retourne vrai si valeur1 est supérieure à valeur2
valeur1 -ge valeur2	Retourne vrai si valeur1 est supérieure ou égale à valeur2

Exemples

Comparez les deux nombres **nombre1** et **nombre2** :

```
trainee@debian:~/formation$ read nombre1
35
trainee@debian:~/formation$ read nombre2
23
trainee@debian:~/formation$ [ $nombre1 -lt $nombre2 ]
trainee@debian:~/formation$ echo $?
1
trainee@debian:~/formation$ [ $nombre2 -lt $nombre1 ]
trainee@debian:~/formation$ echo $?
0
trainee@debian:~/formation$ [ $nombre2 -eq $nombre1 ]
trainee@debian:~/formation$ 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

Exemples

Testez si $fichier n'est pas un répertoire :

```
trainee@debian:~/formation$ fichier=a100
trainee@debian:~/formation$ [ ! -d $fichier ]
trainee@debian:~/formation$ echo $?
0
```

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

```
trainee@debian:~/formation$ repertoire=/usr
trainee@debian:~/formation$ [ -d $repertoire -a -x $repertoire ]
trainee@debian:~/formation$ echo $?
0
```

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

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

Tests d'environnement utilisateur

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

Exemple

```
trainee@debian:~/formation$ [ -o allexport ]
trainee@debian:~/formation$ 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
chaine = modele	Retourne vrai si chaîne correspond au modèle
chaine != modele	Retourne vrai si chaîne ne correspond pas au modèle
chaine1 < chaine2	Retourne vrai si chaîne1 est lexicographiquement avant chaîne2
chaine1 > chaine2	Retourne vrai si chaîne1 est lexicographiquement après chaîne2

Exemple

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

```
trainee@debian:~/formation$[[ -w a100 && ( -d /usr || -d /tmp ) ]]
trainee@debian:~/formation$ 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

Exemples

```
trainee@debian:~/formation$ [[ -d /root ]] && echo "Répertoire root existe"
Répertoire root existe
trainee@debian:~/formation$ [[ -d /root ]] || echo "Répertoire root existe"
trainee@debian:~/formation$
```

L'arithmétique

La commande expr

La commande **expr** prend la forme :

expr Espace nombre1 Espace *opérateur* Espace nombre2 Entrée

ou

expr Tab nombre1 Tab *opérateur* Tab nombre2 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

Exemples

Ajoutez 2 à la valeur de $x :

```
trainee@debian:~/formation$ x=2
trainee@debian:~/formation$ expr $x + 2
4
```

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

```
trainee@debian:~/formation$ expr $x+2
2+2
```

Les opérateurs doivent être protégés :

```
trainee@debian:~/formation$ expr $x * 2
expr: erreur de syntaxe
trainee@debian:~/formation$ expr $x \* 2
4
```

Mettez le résultat d'un calcul dans une variable :

```
trainee@debian:~/formation$ resultat=`expr $x + 10`
trainee@debian:~/formation$ 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

Exemples

```
trainee@debian:~/formation$ x=2
trainee@debian:~/formation$ ((x=$x+10))
trainee@debian:~/formation$ echo $x
12
trainee@debian:~/formation$ ((x=$x+20))
trainee@debian:~/formation$ 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
```

Exemples

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

<note 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. </note>

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

```
MAX_ACCOUNTS=100
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 Debian, le système exécute le script ~/.profile qui exécute le script ~/.bashrc qui appelle le script /etc/bash.bashrc.

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**.

T.P.#1

<note> En utilisant vos connaissances acquises dans cette unité, expliquez les scripts **~/.profile** et **~/.bashrc** ligne par ligne. </note>

~/.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

```
# System-wide .bashrc file for interactive bash(1) shells.

# To enable the settings / commands in this file for login shells as well,
# this file has to be sourced in /etc/profile.
```

```
# If not running interactively, don't do anything
[ -z "$PS1" ] && return

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

# 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, overwrite the one in /etc/profile)
PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '

# Commented out, don't overwrite xterm -T "title" -n "icontitle" by default.
# If this is an xterm set the title to user@host:dir
#case "$TERM" in
#xterm*|rxvt*)
#    PROMPT_COMMAND='echo -ne "\033]0;${USER}@${HOSTNAME}: ${PWD}\007"'
#    ;;
#*)
#    ;;
#esac

# enable bash completion in interactive shells
#if [ -f /etc/bash_completion ] && ! shopt -oq posix; then
#    . /etc/bash_completion
#fi

# if the command-not-found package is installed, use it
if [ -x /usr/lib/command-not-found -o -x /usr/share/command-not-found ]; then
    function command_not_found_handle {
            # check because c-n-f could've been removed in the meantime
```

```
                if [ -x /usr/lib/command-not-found ]; then
           /usr/bin/python /usr/lib/command-not-found -- $1
                   return $?
                elif [ -x /usr/share/command-not-found ]; then
           /usr/bin/python /usr/share/command-not-found -- $1
                   return $?
        else
           return 127
        fi
    }
fi
trainee@debian:~/formation$ 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
[ -z "$PS1" ] && return

# don't put duplicate lines in the history. See bash(1) for more options
# don't overwrite GNU Midnight Commander's setting of `ignorespace'.
HISTCONTROL=$HISTCONTROL${HISTCONTROL+:}ignoredups
# ... or force ignoredups and ignorespace
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)

export HISTSIZE=1000

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
```

```
shopt -s checkwinsize

# 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) 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

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

# 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 [ -f /etc/bash_completion ] && ! shopt -oq posix; then
    . /etc/bash_completion
fi
```

~~DISCUSSION:off~~

Donner votre Avis

{(rater>id=debian_6_l105|name=cette page|type=rate|trace=user|tracedetails=1)}