Version : **2024.01**

Dernière mise-à-jour : 2024/09/30 13:53

RH12405 - La Ligne de Commande

Contenu du Module

- **RH12405 - La Ligne de Commande**
  - Contenu du Module
  - Le Shell
  - LAB #1 - Le Shell /bin/bash
    - 1.1 - Les Commandes Internes et Externes au shell
    - 1.2 - Les alias
    - 1.3 - Définir le Shell d'un Utilisateur
    - 1.4 - Le Prompt
    - 1.5 - Rappeler des Commandes
    - 1.6 - Générer les fins de noms de fichiers
    - 1.7 - Le shell interactif
      - Caractère *
      - Caractère ?
      - Caractères [ ]
    - 1.8 - L'option extglob
      - ?(expression)
      - *(expression)
      - +(expression)
      - @(expression)
      - !(expression)
      - Caractères d'Échappement
  - 1.9 - Codes Retour
  - 1.10 - Redirections

- 1.11 - Pipes
- 1.12 - Substitutions de Commandes
- 1.13 - Chaînage de Commandes
- 1.14 - Affichage des variables du shell
  - Les variables principales
  - Les Variables de Régionalisation et d'Internationalisation
  - Les variables spéciales
- 1.15 - La Commande env
- 1.16 - Options du Shell Bash
  - Exemples
    - noclobber
    - noglob
    - nounset

Le Shell

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

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

Shell	**Nom**	**Date de Sortie**	**Inventeur**	**Commande**	**Commentaires**
tsh	Thompson Shell	1971	Ken Thompson	sh	Le premier shell
sh	Bourne Shell	1977	Stephen Bourne	sh	Le shell commun à tous les Unix. Sous RHEL 9 : /usr/bin/sh
csh	C-Shell	1978	Bill Joy	csh	Le shell BSD. Sous RHEL 9: /usr/bin/csh
tcsh	Tenex C-Shell	1979	Ken Greer	tcsh	Un dérivé du shell csh. Sous RHEL 9 : /usr/bin/tcsh
ksh	Korn Shell	1980	David Korn	ksh	Uniquement libre depuis 2005. Sous RHEL 9 : /usr/bin/ksh
bash	Bourne Again Shell	1987	Brian Fox	bash	Le shell par défaut de Linux et de MacOS X. Sous RHEL 9 : /usr/bin/bash
zsh	Z Shell	1990	Paul Falstad	zsh	Zsh est plutôt orienté pour l'interactivité avec l'utilisateur. Sous RHEL 9 : /usr/bin/zsh

Sous RHEL 9 le shell **/bin/sh** est un lien symbolique vers **/bin/bash** :

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

LAB #1 - Le Shell /bin/bash

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

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

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

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

1.1 - 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@redhat9 ~]$ 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@redhat9 ~]$ type ifconfig
ifconfig is /usr/sbin/ifconfig
```

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

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

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

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

```
drwxr-xr-x. 2 trainee trainee   6 Oct 19  2023 Templates
drwxr-xr-x. 2 trainee trainee   6 Oct 19  2023 Videos
-rw-r--r--. 1 trainee trainee 442 Sep 25 14:24 vitext
-rw-r--r--. 1 trainee trainee   0 Sep 25 15:11 xyz
```

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

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

```
[trainee@redhat9 ~]$ alias
alias dir='ls -l'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias xzegrep='xzegrep --color=auto'
alias xzfgrep='xzfgrep --color=auto'
alias xzgrep='xzgrep --color=auto'
alias zegrep='zegrep --color=auto'
alias zfgrep='zfgrep --color=auto'
alias zgrep='zgrep --color=auto'
```

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

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

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

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

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

1.3 - Définir le Shell d'un Utilisateur

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

```
[trainee@redhat9 ~]$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:65534:65534:Kernel Overflow User:/:/sbin/nologin
systemd-coredump:x:999:997:systemd Core Dumper:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:998:996:User for polkitd:/:/sbin/nologin
avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin
tss:x:59:59:Account used for TPM access:/dev/null:/sbin/nologin
```

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

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

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

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

Important : Notez que sous RHEL 9, le système nous informe que le shell courant de l'utiisateur **trainee** est **/bin/bash** et non **/usr/bin/bash**. Ceci est du au fait que le répertoire /bin est un lien symbolique pointant vers le répertoire /usr/bin.

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

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

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

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

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

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

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

Important : Vous noterez que le shell actif est toujours **/bin/bash** tandis que le shell stipulé dans le fichier /etc/passwd est le **/bin/sh**. Le shell **/bin/sh** ne deviendra le shell actif de **trainee** que lors de sa prochaine connexion au système.

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

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

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

1.4 - Le Prompt

Le prompt d'un utilisateur dépend de son statut :

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

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

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

Important: L'historique est spécifique à chaque utilisateur.

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

Caractère de Contrôle	Définition
[CTRL]-[P] (= flèche vers le haut)	Rappelle la commande précédente
[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@redhat9 ~]$ ls
```

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

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

```
[trainee@redhat9 ~]$ history
    1  su -
    2  exit
    3  su -
    4  clear
    5  cd /
    6  ls -l
    7  cd afs
    8  ls
    9  cd /
   10  su -
...
   85  echo $SHELL
   86  cat /etc/passwd | grep trainee
   87  chsh
   88  history | more
   89  clear
   90  ls
   91  history
[trainee@redhat9 ~]$ !90
ls
aac  abc  bca  Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos  vitext  xyz
```

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

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

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

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

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

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

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

1.6 - 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@redhat9 ~]$ ls .bash
.bash_history  .bash_logout   .bash_profile  .bashrc
```

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

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

1.7 - 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 …

Caractère Spéciaux	Description
+(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@redhat9 ~]$ mkdir training
[trainee@redhat9 ~]$ cd training
[trainee@redhat9 training]$ touch f1 f2 f3 f4 f5
[trainee@redhat9 training]$ ls
f1  f2  f3  f4  f5
```

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

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

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

Caractère ?

Créez maintenant les fichiers f52 et f62 :

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

Saisissez ensuite la commande suivante :

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

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

Caractères [ ]

L'utilisation peut prendre plusieurs formes différentes :

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

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

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

Ensuite saisissez les commandes suivantes et notez le résultat :

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

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

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

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

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

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

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

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

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

Important : Notez que dans ce cas, il n'existe pas de fichiers dénommés **a**, **b**, **c**, **d**, **e** ou **f**. Pour cette raison, n'ayant trouvé aucune correspondance entre le filtre utilisé et les objets

dans le répertoire courant, le commande **echo** retourne le filtre passé en argument, c'est-à-dire **[a-f]**.

1.8 - L'option extglob

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

```
[trainee@redhat9 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@redhat9 training]$ shopt
autocd          off
assoc_expand_once       off
cdable_vars     off
cdspell         off
checkhash       off
checkjobs       off
checkwinsize    on
cmdhist         on
compat31        off
compat32        off
compat40        off
compat41        off
compat42        off
compat43        off
compat44        off
complete_fullquote      on
direxpand       off
dirspell        off
```

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

```
syslog_history  off
xpg_echo        off
```

?(expression)

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

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

Saisissez la commande suivante :

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

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

***(expression)**

Saisissez la commande suivante :

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

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

+(expression)

Saisissez la commande suivante :

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

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

@(expression)

Saisissez la commande suivante :

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

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

!(expression)

Saisissez la commande suivante :

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

```
f123123123.txt  f123123.txt  f.txt
```

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

Caractères d'Échappement

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

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

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

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

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

```
[trainee@redhat9 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@redhat9 training]$ echo \* est un caractère spécial
* est un caractère spécial

[trainee@redhat9 training]$ echo "* est un caractère spécial"
* est un caractère spécial
```

```
[trainee@redhat9 training]$ echo '* est un caractère spécial'
* est un caractère spécial
```

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

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.

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

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

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

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

```
Swap:          5242876           0     5242876
```

De cette façon, la sortie de la commande free sera ajoutée à la fin de votre fichier après les informations de la commande free.

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

Les canaux d'entrées et de sorties sont numérotés :

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

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

```
[trainee@redhat9 training]$ cd ..
[trainee@redhat9 ~]$ rmdir training/ 2>errorlog
[trainee@redhat9 ~]$ cat errorlog
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@redhat9 ~]$ 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 **<** :

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

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

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

1.11 - Pipes

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

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

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

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 :

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

Rappelez-vous que la sortie standard ne peut être redirigée que dans une seule direction. Afin de pouvoir rediriger la sortie standard vers un fichier **et**

la visualiser à l'écran, nous devons utiliser la commande **tee** avec un pipe :

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

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

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

Important : 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.

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

```
Thu Sep 26 12:56:17 PM CEST 2024
```

Important : 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.

1.13 - 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.

1.14 - 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@redhat9 ~]$ echo $LC_ALL

[trainee@redhat9 ~]$ echo $LC_CTYPE

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

```
LC_ADDRESS="en_US.UTF-8"
LC_TELEPHONE="en_US.UTF-8"
LC_MEASUREMENT="en_US.UTF-8"
LC_IDENTIFICATION="en_US.UTF-8"
LC_ALL=
```

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

1.15 - 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@redhat9 ~]$ env
SHELL=/bin/bash
HISTCONTROL=ignoredups
HISTSIZE=1000
HOSTNAME=redhat9.ittraining.loc
PWD=/home/trainee
LOGNAME=trainee
XDG_SESSION_TYPE=tty
MOTD_SHOWN=pam
HOME=/home/trainee
```

```
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=01;37;41
:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*
.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip
=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.b
z2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;
31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.w
im=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bm
p=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01
;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35
:*.webm=01;35:*.webp=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.
wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;
35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=01;36:*.au=01;36:*.f
lac=01;36:*.m4a=01;36:*.mid=01;36:*.midi=01;36:*.mka=01;36:*.mp3=01;36:*.mpc=01;36:*.ogg=01;36:*.ra=01;36:*.wav=0
1;36:*.oga=01;36:*.opus=01;36:*.spx=01;36:*.xspf=01;36:
SSH_CONNECTION=10.0.2.1 37578 10.0.2.101 22
XDG_SESSION_CLASS=user
SELINUX_ROLE_REQUESTED=
TERM=xterm-256color
LESSOPEN=||/usr/bin/lesspipe.sh %s
USER=trainee
SELINUX_USE_CURRENT_RANGE=
SHLVL=1
XDG_SESSION_ID=4
XDG_RUNTIME_DIR=/run/user/1000
SSH_CLIENT=10.0.2.1 37578 22
which_declare=declare -f
XDG_DATA_DIRS=/home/trainee/.local/share/flatpak/exports/share:/var/lib/flatpak/exports/share:/usr/local/share:/u
sr/share
PATH=/home/trainee/.local/bin:/home/trainee/bin:/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin
SELINUX_LEVEL_REQUESTED=
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
MAIL=/var/spool/mail/trainee
SSH_TTY=/dev/pts/0
```

```
BASH_FUNC_which%%=() {  ( alias;
 eval ${which_declare} ) | /usr/bin/which --tty-only --read-alias --read-functions --show-tilde --show-dot $@
}
_=/usr/bin/env
OLDPWD=/home/trainee/training
```

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
```

1.16 - Options du Shell Bash

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

```
$ set -o [Entrée]
```

Par exemple :

```
[trainee@redhat9 ~]$ set -o
allexport       off
braceexpand     on
emacs           on
errexit         off
errtrace        off
functrace       off
hashall         on
histexpand      on
history         on
ignoreeof       off
interactive-comments    on
keyword         off
```

```
monitor         on
noclobber       off
noexec          off
noglob          off
nolog           off
notify          off
nounset         off
onecmd          off
physical        off
pipefail        off
posix           off
privileged      off
verbose         off
vi              off
xtrace          off
```

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

```
[trainee@redhat9 ~]$ set -o allexport
[trainee@redhat9 ~]$ set -o
allexport       on
braceexpand     on
emacs           on
errexit         off
errtrace        off
functrace       off
hashall         on
histexpand      on
history         on
ignoreeof       off
interactive-comments    on
keyword         off
monitor         on
noclobber       off
```

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

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@redhat9 ~]$ set +o allexport
[trainee@redhat9 ~]$ set -o
allexport       off
braceexpand     on
emacs           on
errexit         off
errtrace        off
functrace       off
hashall         on
histexpand      on
history         on
ignoreeof       off
interactive-comments    on
keyword         off
```

```
monitor         on
noclobber       off
noexec          off
noglob          off
nolog           off
notify          off
nounset         off
onecmd          off
physical        off
pipefail        off
posix           off
privileged      off
verbose         off
vi              off
xtrace          off
```

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

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

noglob

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

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

nounset

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

[trainee@redhat9 ~]$
```

Important : 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.

Copyright © 2024 Hugh Norris.