# Bash : Vision d'ensemble

- Objet : Début de la série de wiki visant à maîtriser bash par les caractère.
- Niveau requis :
  débutant, avisé
- Commentaires : Bash, ligne de commande et scripts
- Suivi :
  en-chantier
  - Création par Hypathie 20/03/2014
  - Testé par Hypathie en Avril 2014
- Commentaires sur le forum : ici[1)]

**Nota :** Contributeurs, les Fix Me! sont là pour vous aider, supprimez-les une fois le problème corrigé ou le champ rempli !

Page en court de réécriture



## Introduction

Cette suite de pages wiki se propose, de nous guider dans l'apprentissage de l'interpréteur de commande **bash** (Bourn Again SHell).
Nous l'explorons à travers les caractères et opérateurs, qu'il met à notre disposition. Ils permettent de structurer les commandes pour manipuler nos systèmes GNU/Linux.

Accessible à un niveau débutant,
Cette suite de document à pour but, de dépasser ce stade, et de nous mener à un niveau intermédiaire (avisé).

Plusieurs voies d'approches, peuvent apporter compléments et meilleur compréhension.
Ainsi nous pouvons aussi lire, les autres pages du wiki que propose DF :

- Le shell pour tous
- shell bash

C'est pages ne sont pas redondantes. Elles sont complémentaires.

Il est évident qu'un tel apprentissage, ne peut se faire sans quelques notions de bases informatiques. Présentées plus bas, dans la suite de cette page.
Ainsi, que sans, la terminologie que bash nous impose. Indispensables pour la compréhension de sa syntaxe et de sa grammaire.

Les terminologies de bash, peuvent différer d'autres langages.
Ce n'est pas pour se démarquer, que les concepteurs et documentalistes ont choisi d'autres termes ou d'autres sens à des termes plus usuels.
Mais pour différencier correctement les différents éléments, afin de limiter les confusions.
Ainsi nous pouvons dire que bash à du caractères ;).

Caractères,
Par ce que bash utilise dans sa syntaxe, beaucoup de caractère et peut de mots (nommés mots réservés) pour appeler ses fonctionnalités.
Quoi que, comme nous le verrons plus loin, un simple caractère peut être un mot.

**Yep ! C'est parti !** 😎

# Le shell

Un **shell**, nommé aussi **interpréteur de commande**, est un logiciel en espace utilisateur qui permet d'ordonner des actions au système d'exploitation.

De manière **interactif**, le shell permet à l'utilisateur de saisir, une ligne de commande au clavier, qui après interprétation, est exécutée.
Lorsque le shell exécute une commande, sans précision, ni action supplémentaire, le shell attend la fin de l'exécution de la commande avant de redonner la main à l'utilisateur.

Un shell est capable d'exécuter par lot, un ensemble de commande. La main est alors rendu à l'utilisateur, après l'exécution de la dernière commande du lot. Un lot de commande peut être écrit dans un fichier que nous nommons **script**.

Un shell, peut de manière **non-interactif** exécuter un script. Lorsque toutes les commandes contenues dans le script sont exécutées, le shell se termine, sans donner la main à l'utilisateur.


Si un script est exécuté de manière non-interactif par un shell. Et que le script comporte des commandes qui attendent une interaction avec l'utilisateur. Ce n'est pas le shell qui n'est pas non-interactif, mais le script lui même.

Un shell dispose d'un langage de programmation, offrant la gestion de variable (déclaration, affectation, modification et suppression). L'écriture de boucle répétitive, de structure de choix, et de structure d'exécution conditionnelle.

Que le shell soit exécuté de manière interactif ou pas, il permet :

- D'accéder aux **système de fichier**. [2)]
- De fournir un environnement de travail, constitué de donnée. Mémorisée, sous la forme d'un

couple nom/valeur, nommé variable. L'ensemble de ces variables sont nommées **variables de l'interpréteur**. S'ajoutent les paramètres positionnels (ou variables spéciales en csh) et autres paramètres spéciaux.
- De fournir un environnement, nommé **environnement**, constitué d'une sélection de variable, parmi les variables de l'interpréteur. L'**environnement** est un composant de l'environnent d'exécution.
- De fournir un **environnement d'exécution**, qui rassemble les informations nommées données d'un processus, indispensables au fonctionnement des processus. L'**environnement d'exécution** est transmis aux **processus fils**[3] créés par le shell.
- D'exécuter des programmes dans des environnements, que nous nommons **processus**. Chaque programme dispose de son propre **processus**. Le shell, ne peut créer que des **processus fils**, nommés aussi **sous-shell** (**subshell**).
- De **suspendre**, **reprendre** ou **terminer** (tuer) l'exécution d'un processus en lui **envoyant un signal**.
- De **rediriger** :
  - L'**entrée standard** depuis le clavier, la sortie standard ou un fichier.
  - Les **sorties standards**[4] vers l'écran ou des fichiers.
- De réaliser une **canalisation** (**pipeline**) entre deux processus. Les sorties standards d'un processus sont connectées avec l'entrée standard d'un autre processus. Quand le premier processus n'envoie rien vers ses sorties standards, l'exécution du second processus est suspendu.
- De remplacer le programme exécuté dans un processus par un autre programme.
- La **gestion des tâches**, en passant un processus en **arrière plan**, ou de le remettre au **premier plan**.
- De récupérer l'état final d'une commande, nommé **code de retour**.
- De regrouper des commandes, nommé **lot de commande** ou **suite de commande**.

Certains shells sont dit restreint. Ils restreignent pour des raisons de sécurité et d'optimisation, certaines fonctionnalités.

Exécuté de manière interactif, un shell offre des fonctionnalités supplémentaires :

- Il permet d'interpréter une ligne saisie directement au clavier.
- D'afficher sur l'écran, si elles ne sont par redirigées, les sorties standards des commandes exécutées.
- De redonner la main à l'utilisateur, après l'exécution au premier plan, d'une commande ou d'un lot de commande.
- Il offre un environnement de travail à l'utilisateur :
  - Un **prompt** (ou invite) est affiché lorsque le shell rend la main à l'utilisateur.
  - Un outil d'édition qui permet de naviguer dans la ligne en court de saisie.
  - Certains shells dispose :
    - D'un **historique** des commandes déjà exécutées. Ainsi une commande déjà utilisée peut-être rappelée sur la ligne en court de saisie.
    - D'une **complétion** des commandes, afin de faciliter la saisie des commandes.

Il existe de nombreux shells qui ont été développés au fil du temps.
**Bash** est l'un des shells disponibles, le plus repandu, installé par défaut sur de nombreux systèmes, dont Debian GNU/Linux.
Le Bourn-again shell (bash) est une implémentation libre du Bourn shell (l'un des premiers shell UNIX).
Il a été développé par la **free software fondation**, pour le projet **GNU**. afin d'avoir un shell libre pour les UNIX libre.

**Voir aussi :**

- (gnu)(en) Bash Reference Manual : What is a shell ?
- Historique des shells Unix et GNU/Linux
  - (fr.wikipedia) Shell Unix : Historique des shell Unix
  - (developer.ibm)(en) Evolution of shells in Linux
- Liste des shells
  - (fr.wikipedia) Shell Unix : shell
  - (packages.debian)Paquets logiciels dans « bullseye », Sous-section shells

# Consoles et terminaux

Un shell interactif utilise une interface de ligne de commande (CLI : Commande Line Interface), aussi nommée interface en mode texte.

Pour différencier les différents types de CLI qui sont mises à notre disposition, nous utilisons communément deux termes :

- **La console** qui désigne, les interfaces que nous obtenons par la combinaison des touche `Ctrl+Alt+F…`
- **Le terminal** pour désigner, dans un environnent graphique, la fenêtre où nous obtenons une interface en mode texte.

Cela est simple et précis.
Mais lorsque nous souhaitons aller plus loin, c'est finalement réducteur et porte à confusion.

Pour retrouver les définitions, voir : Terminaux et consoles, explications.

---

**Sur un système non graphique**, nous disposons uniquement d'interface en mode texte.

Sur un système Debian GNU/Linux moderne, 6 consoles sont disponibles.

Elles sont accessibles par l'usage des touches `Ctrl+Alt+F1` à F6.

Leur usage est sanctionné par la saisi au clavier d'**un login** (couple nom de l'utilisateur + mot de passe).

```
Debian GNU/Linux 11 deb11pc

deb11pc login:_
```

Lors de la saisie du mot de passe, aucun caractère n'est affiché.

→ Ainsi personne ne peut lire le mot de passe durant sa saisie.

---

**Sur un système graphique**, lorsque qu'il est démarré,
Nous nous trouvons directement sur une 7em console (pour les système Debian GNU/Linux moderne).

Un login graphique nommé **gestionnaire de session graphique** ou **gestionnaire d'affichage** (**X display manager**) est affiché.

Pour continuer, nous devons renseigner notre nom d'utilisateur, puis notre mot de passe.
Une fois connecté, le bureau est affiché à l'écran.

Pour disposer d'une interface en mode texte (et accéder à un shell interactif), nous devons exécuter une application nommée un **terminal fenêtre** ou **émulateur de terminal pour X**, plus communément **terminal**.

Le shell est directement disponible, aucun login n'est demandé. Par défaut, nous sommes connectés avec notre nom d'utilisateur.

Nous pouvons, quand nous le souhaitons, changer d'utilisateur avec la commande **su**.

Quand nous naviguons entre les différentes consoles, pour revenir sur l'interface graphique,
La combinaison `Ctrl+Alt+F7` (ou la commande **chvt 7**) doit être utilisée.

---

**Voir aussi** :

- console
- terminal

# Shell BASH

## Connaître votre shell utilisateur

### En listant les variables d'environnement

La commande env permet de lister les variables d'environnement du contexte qui l'exécute.

```
env
```

retour de la commande

```
SHELL=/bin/bash
```

> Le retour est copieux !
> Mais on trouve dans la liste la ligne ci-dessus
> signifiant que le programme associé à la variable SHELL est le bash ;
> autrement dit, que le nom de l'interpréteur de commande est bash.

#### En affichant la variable SHELL

Plus directement, on peut faire :

```
echo $SHELL
```

retour de la commande

```
/bin/bash
```

#### Version utilisée

Pour connaître la version de votre shell Bash, tapez :

```
bash --version
```

retour de la commande

```
GNU bash, version 3.2.39(1)-release (i486-pc-linux-gnu)
Copyright (C) 2007 Free Software Foundation, Inc.
```

#### En savoir plus

Avant d'aller plus loin :

- Le shell pour tous
- Et qui dit shell, dit commandes : Utiliser GNU/Linux en ligne de commande, tout commence là !. 

## Commandes internes et externes

Distinguons maintenant les commandes internes et les commandes externes au shell bash,

commandes simples et les commandes composées.

### Les commandes internes

Une commande interne est une commande dont le code est implémenté au sein même du shell. Les commandes sont intégrées, soit pour des raisons de performances (l'appel d'une telle commande ne crée pas de processus fils[5] du shell courant); soit parce qu'une commande intégrée se sert des variables internes du shell.

Cela signifie que lorsqu'on change de shell courant (par exemple bash, dash, zsh ou C-shell[6], on ne dispose plus des mêmes commandes internes.

Néanmoins, les commandes courantes qui sont essentielles à l'utilisateur, se retrouvent sous les différents shell des distributions Linux (le tronc commun standardisé respectant en général la norme POSIX).

- Pour afficher la liste des commandes internes bash et leur syntaxe, la commande :

```
help
```

- Pour afficher une aide sommaire sur une commande interne :

```
help nom_commande
```

### Les commandes externes

Une commande externe est une commande dont le code se trouve dans un fichier exécutable séparé.

- Pour connaître la liste des commandes installées sur son système, on peut lister le contenu des dossiers suivants :
  - `/bin/`[7]
  - `/sbin/`[8])
  - `/usr/bin/` et `/usr/sbin/`[9]
- Pour afficher le chemin d'une commande ainsi que celui de sa page man :

```
whereis nom_commande
```

- Pour afficher simplement son chemin :

```
which nom_commande
```

- Le shell crée un *processus* pour exécuter une commande externe. Parmi les commandes externes que l'on trouve dans un système, il y a les exécutables

ELF (ex. ls, mkdir, vi, sleep) et les fichiers de scripts (dont par exemple les scripts shell).

### Localisation des commandes

La localisation du code d'une commande externe doit être connu du shell pour qu'il puisse exécuter cette commande. A cette fin, bash utilise la valeur de sa variable prédéfinie PATH.

**Pour connaître le statut d'une commande, avec bash, on peut utiliser la commande interne type**:

```
type cd
```

retour de la commande

```
cd est une primitive du shell
```

```
type cp
```

retour de la commande

```
cp est /bin/cp
```

```
type sleep
```

retour de la commande

```
sleep est /bin/sleep
```

> `/bin/commande` signifie donc que c'est une commande externe.

```
type ls
```

retour de la commande

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

```
whereis ls
```

retour de la commande

```
ls: /bin/ls /usr/share/man/man1/ls.1.gz
```

```
which ls
```

retour de la commande

```
/bin/ls
```

> `ls` est donc l'alias de la commande externe `/bin/ls`

## Compositions de commandes

### Les commandes simples

Les commandes simples peuvent être des commandes internes ou des commandes externes.

1. commandes internes par exemple : type, cd , echo , pwd, export ...
2. commandes externes par exemple : ls, mkdir, rm, rmdir, vi, cal ...

- Voici une liste non exhaustive des commandes simples, on y retrouve des commandes internes et externes :

```
cat, chgrp, chmod, chown, cp, date, dd, df, dmesg, echo, ed, export,
false, kill, ln, login, ls, mkdir, mknod, more, mount, mv, ps, pwd,
rm, rmdir, sed, setserial, sh, stty, su, sync, true, umount, uname.
```

> Consultez la documentation pour plus d'information sur chacune d'entre elles.

### Les commandes composées par des mots clés

Les commandes composées peuvent toutes être considérées comme des commandes internes, en tant qu'elles sont des structures de contrôle.

- Voici les commandes composées :

```
case ... esac ; if ... fi ; for ... done ; select ... done ; until ... done
; while ... done ;
{...} ; ( ... ) ; ((...)) ; [ ... ] ; [[ ]]
```

- Sur l'utilisation et la syntaxe de ces commandes voir : Fonctionnalités avancées du Shell

**Remarque** : Un mot clé est un mot, une expression ou un opérateur réservé. Il a une

signification particulière pour le shell. Un ensemble de mots clés constitue un bloc permettant la syntaxe du shell. Un mot clé n'est pas strictement une commande, mais fait partie d'un ensemble plus large de commandes.

## Notion de processus

### Tout est là

- processus
- la commande ps
- la commande top
- http://fr.wikibooks.org/wiki/Le_syst%C3%A8me_d%27exploitation_GNU-Linux/Processus

### Rappel : deux façons de lister les processus

- Pour se faire plaisir en observant la hiérarchie des processus :

```
pstree -p
```

> l'option `-p` permet d'afficher le PID en plus de la hiérarchie.

- Pour se faire très plaisir en observant la liste des processus de manière dynamique :

```
top
```

### Exemple pratique : tuer un processus

Imaginons que vous vous êtes endormi devant votre messagerie.
A votre réveil : impossible de fermer la fenêtre de icedove avec la souris.
Dans ce cas, le plus simple est tuer le processus d'exécution de icedove.

- Retrouver le processus :

1. ouvrir par exemple tty1 : Ctrl+Alt+F1
2. se loguer et entrer son mot de passe utilisateur
3. récupérer le PID du processus icedove :

```
ps -A | grep icedove
```

retour de la commande

```
4245  pts/0   00:00:02  icedove
```

> Ici 4245 est le PID du processus.

> Il ne sera pas le même si vous exécutez cette commande sur votre machine, ou d'une fois à l'autre.

- Tuer le processus :

```
kill 4245
```

```
exit
```

> Mieux prendre l'habitude de ne pas laisser une console `tty` sans surveillance après s'y être loguer, et cela d'autant plus pour le compte root -;)

- Revenir sur l'interface graphique : Alt+F7

Quelques exemples pour illustrée la notion de processus :

- Exportation de la valeur d'une variable.
- sous-shell

# Quotes, apostrophe, guillemets et apostrophe inversée

## Simple quote ou apostrophe

Les simples quotes : ' délimitent une chaîne de caractères.

Même si cette chaîne contient des commandes ou des variables shell, celles-ci ne seront pas interprétées. Par exemple :

```
variable='secret'
echo 'Mon mot de passe est $variable.'
```

retour de la commande

```
Mon mot de passe est $variable.
```

## Doubles quotes ou guillements

Les doubles quotes : “ délimitent une chaîne de caractères, mais les noms de variable sont interprétés par le shell. Par exemple :

```
variable="secret"
echo "Mon mot de passe est $variable."
```

retour de la commande

```
Mon mot de passe est secret.
```

Ceci est utile pour générer des messages dynamiques au sein d'un script.

**Remarquez bien la différence :**

- Comme ceci, le shell va se trouver à interpréter **chaque argument séparément** :

```
echo coucou tout le monde
```

- Comme cela le shell interprète toute la chaîne **comme un seul argument**.

```
echo "coucou tout le monde"
```

## Anti-quote ou apostrophe inversée

Bash considère que les anti-quotes (`) délimitent une commande à exécuter.

Les noms de variable et les commandes entre ` sont donc interprétés, et remplacées par la sortie de ces commandes. Autrement dit, les anti-quotes[10)] remplacent de manière itérative un argument par une commande, comme le fait la commande xargs.

### Préparation

- soit un dossier `Dossier.txt` contenant les fichiers `dossier1`; `dossier2` ; `dossier3`.

```
cd /tmp
```

```
mkdir Dossier.txt
```

```
cd Dossier.txt/
```

```
touch dossier1
```

```
touch dossier2
```

```
touch dossier3
```

- toujours au niveau de Dossier.txt :

```
ls
```

retour de la commande

```
dossier1  dossier2  dossier3
```

```
rm `ls`
```

Attention, cette commande est dangereuse !
Elle efface tout le contenu du dossier.
À ne pas lancer dans votre dossier personnel sous peine perdre irrémédiablement tous vos fichiers !

```
ls
```

> Pour vérifier que tout a été supprimé :
>
> - comparez maintenant avec :

```
touch dossier1
```

```
touch dossier2
```

```
touch dossier3
```

```
ls
```

retour de la commande

```
dossier1  dossier2  dossier3
```

```
ls | xargs rm
```

```
ls
```

> Pour vérifier que tout a été supprimé :
>
> - Autre exemple :

```
echo `ls`
```

> Cette commande affiche le contenu du répertoire courant à l'écran.
> Elle est (presque) équivalente à ls.

- **À voir** : Guide avancé d'écriture des scripts Bash: 11. Substitution de commandes

# Métacaractères et "métacaractères" !

## Définition usuelle de métacaractère et détail

- Un métacaractère (en anglais, wild card ou joker) est un caractère qui représente un ou

plusieurs autres caractères qui, eux, sont interprétés littéralement.

Certains caractères spéciaux sont appelés *métacaractères* ; soit parce qu'ils servent à effectuer des recherches sur les mots ; soient parce qu'ils servent dans les expressions rationnelles; soit encore parce qu'ils représentent symboliquement quelque chose, un fichier, la valeur d'une variable ; finalement parce que ces caractères représentes symboliquement quelque chose;

- D'une part, le wiki "les métacaractères, ou globs, ou encore patterns" présente les métacaractères `?`, `;`, `*` et les crochets `[ ]` qui sont communs aux différents shell.

Voici un tableau qui regroupe les métacaractères :

| **Communs à différents shell** | |
|---|---|
| stricts | ? (pour un caractère) |
| | * (plusieurs, ou aucun, caractères) |
| brackets | [ ] (plage de caractères [12] ou union [1-3]) |

- D'autre part, man glob et man bash dénomme *caractères génériques*[11], ce que nous appelons des métacaractères .
  Attention à la confusion ! Voir plus bas définition particulière de man bash.
- Quant au globbing, cela concerne l'opération qui permet d'invoquer, par un motif générique, une liste de noms de fichiers, pouvant correspondre à ce motifs.

Pour simplifier, on peut appeler les caractères génériques et expressions génériques, des globs en référence à la commande glob ou à l'option extglob de la commande shopt.
Voir : man bash ligne 1914,sq. et man glob

En définitive, dans la suite j'emploierai le terme métacaractère comme synonyme de globs simples et bracket.

- Enfin, ces expressions génériques (man bash), ou métacaractères (terminologie usuelle) sont issues d'un mécanisme plus large et plus complexe appelé, *expressions rationnelles*[12].
  - Voir : Regexp et grep
  - Voir : Regex et sed
  - Pour un rappel des principaux caractères des expressions rationnelles voir : caractères des regexp étendues.
- Dans un contexte proche, c'est-à-dire qui concerne la gestion des chaînes de caractères, le shell possède des mots réservés et des syntaxes qui permettent de modifier la valeur des variables ou paramètres créés(ées) par l'utilisateur.
- Dans `${ }` on peut trouver les caractères :

`:-` ; `:=` ; `:` ; `::` ; `:+` ; `:?` ; `@` ; `*` ; `##` ; `%%` ; `%`

> Ne pas confondre avec joker ou bracket
> C'est le sujet du tuto : les caractères de transformation de paramètres

## Globs simples, ou métacaractères

- Détails et exercices, l'essentiel est là : métacaractères, ou globs, ou encore patterns.

### Comment associer le point et l'étoile ?

On sait que `ls -A` permet de lister tous les fichiers, le retour est alors un peu trop copieux

De même `ls -a *` est très prolixe.

On peut alors faire :

```
ls -d .*
```

> On obtiendra alors tous les fichiers et dossiers *cachés*.

Pour affiner la recherche et n'avoir que ceux dont le nom commence par un “c”, on peut faire :

```
ls -d .c*
```

retour de la commande

```
.cache  .config
```

Il y a un piège !
Rappel:

1. L'étoile correspond à n'importe quelle chaîne.
1. Si l'on fait `*.txt` on représente tous les fichiers se terminant par `.txt`.
1. Mais attention : `.` représente un point `.` mais aussi deux points `..`

```
ls .*    # et son équivalent : ls .?*
```

> Cela va afficher le répertoire courant (.) ;
> tous les fichiers cachés (normaux) ;
> ainsi que tous les fichiers cachés de type-répertoire :
>
> avec les noms des fichiers cachés contenus dans ces répertoires ;
> mais aussi du répertoire parent (..) ;
> tous les fichiers cachés normaux et les fichiers cachés de type-répertoire,
> avec les noms des fichiers cachés contenus dans ces répertoires !

Ça fait beaucoup trop

Solutions :

```
ls .[!.]*
```

> Cela liste le répertoire courant ;
> tous les fichiers cachés normaux ;
> les fichiers cachés de type-répertoire ainsi que les fichiers cachés de ces répertoires.

```
ls -d .[!.]*   # et son équivalent : ls -d .??*
```

> liste le répertoire courant ;
> tous les noms de fichiers normaux cachés ;
> et le nom de tous les fichiers cachés de type-répertoire (sans leurs contenus cette fois !)

C'est beaucoup mieux

**Vivre dangereusement supprimer tous les fichiers cachés d'un répertoire !**

- Soit un répertoire “Hide-files” contenant plusieurs fichiers cachés:

```
mkdir Hide-files && cd Hide-files && touch ./.fichier1 ./.fichier2
./fichier1 ./fichier2
```

- Pour visualiser ce qu'on fait :

```
ls -la
```

retour de la commande

```
total 8
drwxr-xr-x  2 hypathie hypathie 4096 juil.  8 10:43 .
drwxr-xr-x 34 hypathie hypathie 4096 juil.  8 10:43 ..
-rw-r--r--  1 hypathie hypathie    0 juil.  8 10:43 fichier1
-rw-r--r--  1 hypathie hypathie    0 juil.  8 10:43 .fichier1
-rw-r--r--  1 hypathie hypathie    0 juil.  8 10:43 fichier2
-rw-r--r--  1 hypathie hypathie    0 juil.  8 10:43 .fichier2
```

- Pour en supprimer uniquement les fichiers cachés :

```
rm -i \.f*
```

retour de la commande

```
rm : supprimer fichier vide « .fichier1 » ?
```

```
rm : supprimer fichier vide « .fichier2 » ?
```

> Répondre “yes” et taper ↵ Entrée
>
> L'option `-i` permet de demander une confirmation avant chaque effacement.

- Pour visualiser ce qu'on a fait :

```
ls -la
```

retour de la commande

```
total 8
drwxr-xr-x  2 hypathie hypathie 4096 juil.  8 10:48 .
drwxr-xr-x 34 hypathie hypathie 4096 juil.  8 10:43 ..
-rw-r--r--  1 hypathie hypathie    0 juil.  8 10:43 fichier1
-rw-r--r--  1 hypathie hypathie    0 juil.  8 10:43 fichier2
```

- Avec la commande `rm` :

1. toujours s'assurer que vous êtes dans le répertoire parent des fichiers cachés à supprimer[13)]
2. l'option `-i` pour plus de maîtrise ;
   surtout quand on l'utilise en tant que en root !
3. mieux vaux éviter de la placer sur le slash `/*` (pour aller plus vite)
   il est plus prudent de mettre l'étoile après la première lettre `rm \.f*` quitte à perdre un peu de temps.
4. Attention lors de la rédaction du chemin absolu d'un fichier :

> Il suffit par exemple de mettre par accident un espace après le slash :
>
> `rm / home/user`
>
> et ce pourrait être la catastrophe !

- Évitez toujours: `rm -rf` , surtout avec `/`:

1. l'option `-r` ou `-R`: opère récursivement sur un dossier (=supprime ses sous-dossiers);

1. l'option `-f` : permet de forcer la suppression (lorsqu'un dossier n'est pas vide par exemple)

- À ne pas faire sans comprendre ce qu'il se passe !

Néanmoins, petits curieux, petites curieuses,
sachez qu'en user comme en root, la plupart des systèmes (pas forcément sur tous !) possèdent une sécurité :

```
rm /*
```

retour de la commande

```
rm: impossible de supprimer « /bin »: est un dossier
rm: impossible de supprimer « /boot »: est un dossier
rm: impossible de supprimer « /dev »: est un dossier
```

> Les premières lignes du retour.
>
> Il faut utiliser `–no-preserve-root` pour inhiber cette mesure de sûreté 😵
>
> - Mais la commande fatale à tous les coups est 😵

```
rm -rf /*
```

> Après son exécution le système est effacé, vos données aussi, et vous êtes triste 😢

## "Métacaractères", ou opérateurs de contrôle et de redirection

### Définition particulière

man bash

```
métacaractère
    Un caractère qui, non protégé, sépare les mots. Un de ceux-ci :

        | & ; ( ) < > espace tabulation
```

> Un *mot* est une séquence de caractères considérée comme une unité élémentaire par le shell.
> On parle également de *token* (jeton).

- Attention dans ce sens, le terme de “métacaractère” ne renvoie pas aux caractères génériques (man bash) ou au globbing et pattern mais fait référence à l'analyse et au traitement par le shell de la ligne de commande.
- L'ordre d'analyse est le suivant :

1. découpage lexical en mots ;
2. découpage en commandes (lignes de tubes, instructions de contrôle (if, while...) ) ;
3. analyse des redirections ;
4. expansion des paramètres ;
5. substitution de commande;

6. redécoupage en mots des chaînes substituées;
7. expansion des noms de fichier;
8. lancement des commandes;*
9. récupération de la sortie

En bref !!!

- Les caractères servant au globbing et au patterns, sont désignés dans le man bash de *caractères génériques*.
  Mais à l'usage on parle de métacaractères.
- Les caractères qui permettent le mécanisme de lecture d'une ligne de commande par le shell sont appelés dans le man bash *métacaractères*.
  Mais à l'usage on parle de mots réservés.
- On distingue deux sortes de mots réservés : les opérateurs de contrôle et les opérateurs de redirection.

## Les opérateurs de contrôle

**Parmi tous les opérateurs de contrôle ci-dessous :**

man bash

```
    ||    &&    ;    &    ;;    ( )    |    <retour-chariot>
```

**Il faut premièrement distinguer ceux qui servent à enchaîner les commandes :**

`|| && ; <retour-chariot>`

Voir :

- Le shell pour tous
- enchainer-plusieurs-commandes
- Enchaînements de commandes dans les scripts
- l'enchaînement conditionnel

**Remarques**

Quant aux autres : `& ;; ( ) |`

- Le signe `|` est un “ou” logique.

> Il ne fait pas référence ici au pipe qui sert dans les tubes.
> Ce “ou” logique apparaît dans deux cas :
>
> - le | dans le contexte d'utilisation des globs étendus et des expressions rationnelles
>   voir : bash-vii-globs-etendus-regex
>   Par exemple :
>
> ```
> ls ~/Test/!(*jpg|*bmp)
> ```
>
> - Le | avec les ;; dans contexte de la commande case

```
case chaîne in
      choix1 )  commande ;;
      choix2 | choix3 ...) commande ;;
#             ^
      ...
      * )  commande ... ;;
esac
```

> - On retrouve la paire de parenthèses simples ( ):
>   - Avec la commande function

```
name (){
commands
return $TRUE
}
name
```

> - Avec les substitutions de commande $( )
> - Avec les globs étendus (voir lien ci-dessus) ;
>
> ou dans les expressions rationnelle (voir : bash-vii-globs-etendus-regex
>
> Et à pas confondre avec la double paire de parenthèse ( ( ) ) qui ne figure pas dans la liste et qui sert à faire des calculs
> (voir : page-man-bash-iv-symboles-dans-les-calculs-mathematiques.

## Les opérateurs de redirection

```
<     >     >    |    <<    >>    <&    >&
```

Là encore cette liste réclame une explication.
Les opérateurs de direction sont à strictement parler ceux-ci :

```
    >     >>     <     <<     >&     |
```

Il s'agit bien du pipe cette fois, et étant une redirection un peu différente, on le trouve souvent explicité à part.

```
<&-     <&-
```

Permettent la fermeture de l'entrée standard et de la sortie standard.

- À voir :
  - **rediriger-l-affichage**
  - **exercices avec les chevrons**
  - **redirections dans les scripts**

Enfin, ne sont pas désignés de “métacaractères”, ni d'opérateurs de contrôle :
Tous les caractères spéciaux, c'est-à-dire les symboles auxquels le shell est sensible.
Comme nous le verrons dans cette série de wiki, il s'agit :

- de tous les caractères qui ne servent pas à séparer les mots ou les commandes;
- des caractères qui inhibent la reconnaissance des caractères spéciaux et des métacaractères (“glob” ou “patterns” et “bracket expression”);
- des mots réservés des commandes composées;
- des caractères qui transforment un caractère simple en caractère spécial (par exemple, le tiret devant une lettre, fait reconnaître cette lettre comme une option );
- enfin, des caractères symboliques qui représentent :
  - différentes sortes de fichiers;
  - les variables d'environnement prédéfinies;
  - ou encore les paramètres prédéfinis.

# Scripts et Alias

## les scripts

Un **script** est la rédaction dans un fichier texte d'un ensemble de *commandes* et d'expressions régulières (caractères utilisés symboliquement) orientant les instructions données aux commandes.

- **Pour une initiation au script : debuter-avec-les-scripts-shell-bash**
- **sur “if”, “les boucles”, etc. : Fonctionnalités avancées du Shell**
- **considérations avancées : Rédaction de scripts Shell**
- **Pour les DÉBUTANTS AVISÉS francophile : http://abs.traduc.org/abs-fr/pt01.html**
- **Pour les DÉBUTANTS AVISÉS et anglophile, voir là :http://mywiki.wooledge.org/BashFAQ**

- **À voir aussi : les scripts debian-facile :**
  - [pygtk] Comment faire un notebook avec bouton "fermer"
  - [bash] Réalisation d'un script contenant des Alias

- [bash] Déterminer si un fichier ou répertoire existe
- [bash] Déterminer si un partage existe sur un disque réseau de type Synology
- [bash] Lire un fichier texte ligne par ligne
- [bash] Section d'un paquet debian
- [bash] Dépôt d'un paquet debian
- [bash] Convertir dans plusieurs sous-répertoires des images .png en .jpg

### les alias

- **Avant tout le chapitre : Le shell pour tous : Les Alias**
- **L'essentiel est là : Maîtriser les alias bash**
- **alias et script : Réalisation d'un Script contenant des Alias**

# la suite c'est ICI :

Bash : Détail et caractères

1)
N'hésitez pas à y faire part de vos remarques, succès, améliorations ou échecs !
2)
Bonne maxime à retenir : **Tout est fichier**.
3)
Un **processus fils** est un processus créer par un autre processus, alors nommé **processus parent**.
4)
Par défaut, les sorties standards sont deux : la **sortie standard** et la **sortie d'erreur standard**.
5)
processus
6)
Voir : http://fr.wikipedia.org/wiki/Csh
7)
`ls /bin` affiche la liste des commandes externes essentielles utilisées par le système pendant le démarrage, mais utilisables par tous les utilisateurs.
8)
`ls /sbin` affiche la liste des commandes externes essentielles utilisées par le système pendant le démarrage, et souvent réservées à l'administrateur (root
9)
`ls /usr/bin /usr/sbin` affiche la liste des commandes externes secondaires, non utilisées pendant le démarrage du système, et respectement utilisables par tous ou plutôt réservées à l'administrateur
10)
L'anti-quote s'obtient, sur clavier AZERTY, avec les 2 touches simultanées du clavier : AltGr+7.
11)
Et un enchaînement de motifs génériques est nommée par extension, une expression générique. En effet, une chaîne de caractères ordinaires est un motif. Et un motif contenant un ou plusieurs “caractères génériques” est appelé un “motif générique”.
12)
ou *expressions régulières*, et en anglais *regular expressions* souvent abrégé en *regex* ou *regexp*
13)
Observez le prompt : `user@nom-machine:~/REPERTOIRE-PARENT$`

From:
http://debian-facile.org/ - **Documentation - Wiki**

Permanent link:
**http://debian-facile.org/doc:programmation:shells:bash-les-differents-caracteres-speciaux**

Last update: **01/04/2023 19:17**