

CHAPITRE 3 : INTERPRETEUR DE COMMANDES

1. Le shell

Sous Unix, un **shell** est un **interpréteur de commandes** qui fait office d'interface entre l'utilisateur et le système d'exploitation. Chaque commande saisie par l'utilisateur (ou lue à partir d'un fichier) est syntaxiquement vérifiée puis exécutée.

Les nombreux shells se classent en deux familles :

- la famille C shell (ex : csh, tcsh).
- la famille Bourne shell (ex : sh, bash, ksh.)
- **zsh** est un shell qui contient les caractéristiques de ces deux familles.

Néanmoins, le choix d'utiliser un shell plutôt qu'un autre est essentiellement une affaire de préférence personnelle ou de circonstance. En connaître un, permet d'accéder aisément aux autres. Lorsque l'on utilise le système GNU/Linux (un des nombreux systèmes de la galaxie Unix), le shell par défaut est **bash** (Bourne Again SHell). Ce dernier a été conçu en 1988 par Brian Fox dans le cadre du projet GNU2. Aujourd'hui, les développements de bash sont menés par Chet Ramey.

Un shell possède un double aspect :

- un aspect environnement de travail (historique des commandes, lancement de tâches en arrière plan et contrôle de celles-ci, ...).
- un aspect langage de programmation (création de variables, structures de contrôle, ...).

Avantages

- *langage interprété* : pas de compilation, ni édition de liens (comme en C par exemple).
- *le shell manipule essentiellement des chaînes de caractères* : pas de notion de pointeur, évite les erreurs de typages et pointeurs mal gérés. On raisonne tout le temps en chaîne de caractères.
- *prototypage rapide d'applications (tubes, substitutions de commandes et de variables)* : ce qui permet la construction d'une application par assemblage de commandes préexistantes dans l'environnement Unix.
- *langage « glu »* : il permet de connecter des composants écrits dans des langages différents. Le composant doit être capable :
 - de lire sur l'entrée standard
 - d'accepter des arguments et options éventuels
 - d'écrire ses résultats sur la sortie standard
 - d'écrire les messages d'erreur sur la sortie standard dédiée aux messages d'erreur, et de renvoyer un code décrivant l'état de son exécution (si erreur par exemple).

Inconvénients

- *langage ésotérique* : conçu par et pour des développeurs Unix.
- *l'oubli d'un espace* par exemple peut changer le comportement d'une commande.
- *nombreuses syntaxes* pour faire les mêmes choses (soucis de compatibilité).
- *de nombreux caractères* ont une signification différente selon le contexte.

Connaître son shell ? la commande *ps*



```
elrond ouwayedn 54 % ps
PID TTY          TIME CMD
7331 pts/0        00:00:00 tcsh
7870 pts/0        00:00:00 ps
elrond ouwayedn 55 %
```

2. Syntaxe d'une commande

[chemin/]nom_cmd [option ...] [argument ...]

C'est une suite de mots séparés par un ou plusieurs espaces. Une tabulation est considérée comme un espace. Un mot est une suite de caractères sans espaces. Cependant, plusieurs caractères ont une signification spéciale pour le shell et provoquent la fin d'un mot : ils sont appelés méta-caractères (ex: |, <). Nous les détaillerons plus tard. **Attention Bash distingue les caractères majuscules des caractères minuscules.**

- Le **nom** de la commande est le premier mot.
- Une **option** est introduite par un **tiret** (ex : **-a**) ou dans la syntaxe GNU par deux caractères **tiret** consécutifs (ex : **--version**). Elle précise un fonctionnement particulier de la commande.
- Les **arguments** désignent les objets sur lesquels doit s'exécuter la commande.
- La syntaxe [*elt* ...] signifie que l'élément *elt* est facultatif ou bien présent une ou plusieurs fois. Cette syntaxe ne fait pas partie du shell ; elle est uniquement descriptive (méta-syntaxe).

Exemple : `ls -l RepC RepD` : commande `ls` avec l'option `l` et les arguments `RepC` et `RepD`

Rappel : il faut abuser du man

Deux types de commandes

- *internes* (`cd` , `echo` , `for` , `pwd`) : implantées directement dans le shell. Dépendent donc du shell utilisé.

- *externes (ls, mkdir, vi, sleep)* : commandes qui se trouvent dans un fichier ordinaire. Voir par exemple les dossier **/bin** et **/sbin**. Les commandes externes disponibles dépendent des paquetages installés sur le système.

Comment savoir ? la commande *type*



```
elrond.loria.fr - PuTTY
elrond ouwayedn 52 % bash
bash-3.00$ type echo
echo is a shell builtin
bash-3.00$ type mkdir
mkdir is /bin/mkdir
bash-3.00$
```

Diverses catégories de commandes

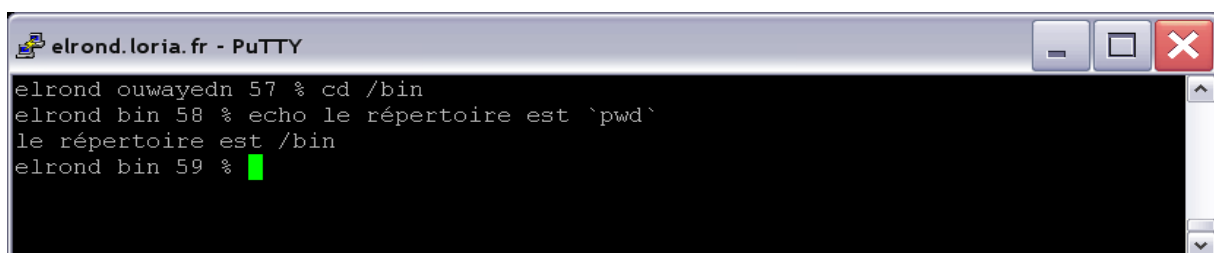
- utilisateurs
- répertoires
- fichiers
- ...

3. Mode d'exécution de commandes

- *simple* : le shell lit la commande entrée par l'utilisateur, l'analyse, la pré-traite et si elle est syntaxiquement correcte, l'exécute.
- *séquentielle* : ensemble de commandes séparées par des « ; » : elles sont exécutées les unes après les autres.
- *en arrière-plan* : permet de lancer une commande et de récupérer immédiatement la main pour lancer "en parallèle" la commande suivante. On utilise le caractère **&** pour lancer une commande en arrière-plan (rappel : parallélisme logique).
- *chaînée (voir processus et redirection de descripteurs, dans les séances suivantes).*

4. Substitution de commandes

Une commande entre apostrophe inverses est remplacée par sa sortie. La ligne de commande est évaluée ensuite.



```
elrond.loria.fr - PuTTY
elrond ouwayedn 57 % cd /bin
elrond bin 58 % echo le répertoire est `pwd`
le répertoire est /bin
elrond bin 59 %
```

5. Caractères génériques et expansion de noms de fichiers (substitution de noms de fichiers)

Les caractères et expressions génériques (voir ci-après) sont issus d'un mécanisme plus général appelé **expressions rationnelles** (*regular expressions*) ou **expressions régulières**. Ces caractères et expressions sont utilisés pour spécifier un *modèle* de noms d'entrées. Ce *modèle* est ensuite interprété par le shell pour créer une liste triée de noms d'entrées. Par défaut, le shell traite uniquement les entrées non cachées du répertoire courant. Cela signifie que les entrées dont le nom commence par le caractère «.» sont ignorées.

Attention : une commande ne reçoit pas de caractères génériques dans ses arguments, mais une liste d'entrées créée par le shell, c.-à-d. l'ensemble des entrées qui vérifient l'expression donnée.

Deux types de caractères :

- caractère **standard** ou **littéral** : a,b,c,d,..., 1,2,3 Il est interprété comme "lui-même".
- **méta-caractère** ou caractère **spécial** : ", \$, ., /, [], *, ?, {}, !, >, <, ^... Il est interprété par le shell et a une signification particulière.

Pour qu'un méta-caractère soit considéré comme un caractère normal, il faut le banaliser.

- Exemple : \ \$ ou \ *

Caractères génériques pour l'expansion de noms de fichiers :

Il existe plusieurs caractères génériques : ce sont des méta-caractères « qui peuvent être remplacés par d'autres ».

- * désigne une chaîne quelconque (même la chaîne vide)
- ? désigne un caractère unique
- [] désigne un seul caractère parmi une liste de caractères
- [!] représente un seul caractère qui n'est pas dans la liste

Exemples d'utilisations :

```
- ls
  f1.c f1.o f1.s f2.c f11.c
- f?.c → f1.c f2.c
- f1.? → f1.c f1.o f1.s
- f?   → f? (aucun)
- f1?.c → f11.c
- f???.c → f11.c
- ?     → ? (aucun)
```

```

- ls
  bob darlene dave ed frank fred
- fr* → frank fred
- *ed → ed fred
- b* → bob
- *e* → darlene dave ed fred
- * → bob darlene dave ed frank fred
- d*e → darlene dave
- g* → g* (aucun)

```

```

[abc] → a, b ou c
[.,:] → ' . ', ' , ' ou ' ; '
[a-c] → a, b ou c
[a-z] → toutes les lettres minuscules
[!0-9] → tout ce qui n'est pas des chiffres
[0-9!] → tous les chiffres et ' ! '
[a-zA-Z] → toutes les lettres (majusc. et minusc.)
[a-zA-Z0-9_-] → toutes les lettres, tous les chiffres
                le souligné et le tiret

```

```

- ls
  f1.c f1.o f1.s f2.c f11.c g1.1 g2.1
- f1.[cs] → f1.c f1.s
- [a-z][0-9].[c1] → f1.c f2.c g1.1 g2.1
- *.[!c] → f1.o f1.s g1.1 g2.1
- *.[!cso] → g1.1 g2.1
- * [!.]? → * [!.]? (aucun)

```

Expansion des accolades :

echo ri{dell,gole,dicule}s

ridells rigoles ridicules

echo b{ar{re,rage},ruit}s

barres barrages bruits

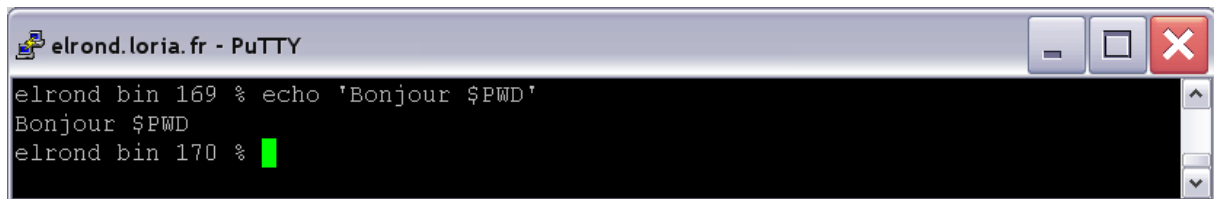
Exercice 1: Tester dans /bin/ l'expression régulière suivante :

- `ls -l ???`

6. Protection de caractères

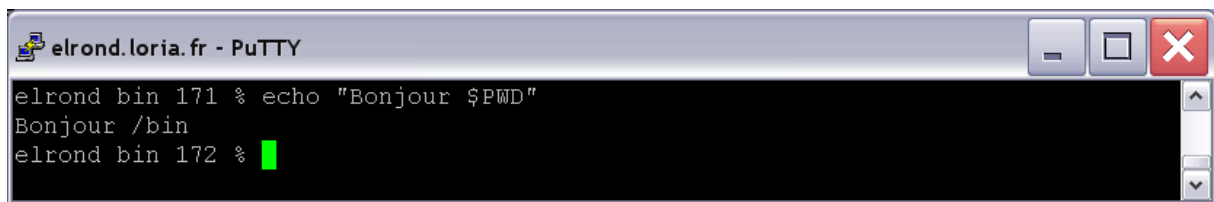
Les caractères spéciaux sont banalisés et donc pas interprétés par le shell s'ils sont protégés.

- Protection forte ou totale : la chaîne est encadrée par des `' '`



```
elrond.loria.fr - PuTTY
elrond bin 169 % echo 'Bonjour $PWD'
Bonjour $PWD
elrond bin 170 %
```

- Protection faible ou partielle : la chaîne est encadrée par des `" "` (expansion partielle) sauf `$\`"`



```
elrond.loria.fr - PuTTY
elrond bin 171 % echo "Bonjour $PWD"
Bonjour /bin
elrond bin 172 %
```

- Protection d'un caractère : `\c`



```
elrond.loria.fr - PuTTY
elrond bin 133 % echo \?
?
elrond bin 134 %
```

7. Expressions régulières

Caractères

- `.` représente un caractère unique
- `[]` un caractère parmi ceux de la liste
- `[!]` un caractère qui n'est pas dans la liste
- `^` début d'expression régulière
- `$` fin d'expression régulière

- `\|` ou logique

Répétitions du caractère précédent

- `?` 0 ou 1
- `*` de 0 à n
- `+` de 1 à n
- `\{x,y\}` de x à y
- `\{x,\}` de x à n
- `\{,y\}` de 0 à y

Sous-expression régulière

- Utilisations de parenthèse `\( \)`, répétable également
- Exemple : `"^\(a|b\).*$"` représente les mots qui commencent par a ou b
- `"ch\(at|ien\)"` correspond à *"chat"* ou *"chien"*

Quelques exemples avec grep

`grep <EXPRESSION> fichier`. Commande qui recherche des expressions régulières dans un fichier et retourne les lignes qui contiennent l'expression.

- `grep abc fichier` recherche la chaîne **abc** dans toutes les lignes du fichier. Les lignes trouvées sont envoyées sur la sortie standard, éventuellement redirigée.
- `grep " " fichier`
- `grep "ab*c" fichier`
- `grep "^s.*n$" fichier`
- `grep "[a-zA-Z][a-zA-Z0-9]*" [a-f]*.txt`