



UNIVERSITÉ DE
MONTPELLIER



Introduction aux systèmes d'exploitation et à leur fonctionnement

Shell

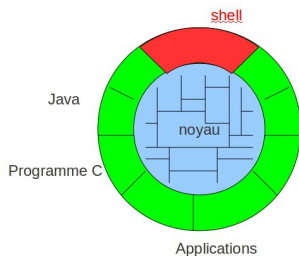
Fabien Laguillaumie (fabien.laguillaumie@umontpellier.fr)

OBJECTIFS DE CETTE PARTIE

Se familiariser avec le *shell* qui est souvent utilisé pour accéder aux services du système en mode commande

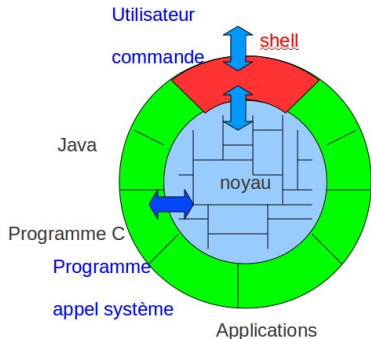
- ▶ Comprendre l'interprétation et l'exécution des commandes
- ▶ Connaître les commandes de base
- ▶ Donner le vocabulaire de base

ARCHITECTURE GLOBALE



- ▶ Dans le **noyau** sont organisés les services de base qui
 - ▶ permettent d'accéder à du matériel, à des informations sur les fichiers, sur des exécutables, etc.
 - ▶ sont souvent accessibles pour les applications
 - ▶ sont écrit en langage C
- ▶ Une application particulière, le **shell**, rend possible l'accès à certains services en mode conversationnel.

ARCHITECTURE GLOBALE



- ▶ Les services peuvent être invoqués
 - ▶ à partir des programmes exécutables (on les utilise sous forme des **appels système**)
 - ▶ à partir du shell en appelant une (des) **commande(s)**
 - ▶ on peut exécuter des commandes les unes après les autres en mode interactif
 - ▶ on peut "programmer" en utilisant les commandes shell en écrivant des scripts

LE SHELL

Le **shell** est une interface simple entre l'utilisateur et le système

- ▶ Plusieurs shells existent
- ▶ Ils peuvent être installés et accessibles dans un même système
 - ▶ GNU Bourne-Again SHell (bash) : shell souvent accessible
 - ▶ /bin/bash
 - ▶ /bin/sh est un lien vers /bin/bash
 - ▶ C shell
 - ▶ K shell
 - ▶ Z shell

LES COMMANDES SHELL

- ▶ On a déjà vu des commandes de base :
ls, cd, pwd, rm, chmod, man,...

- ▶ Syntaxe générale :

commande [OPTIONS] [ARGUMENTS, NOMS]

LES MÉTA-CARACTÈRES

Les méta-caractères

- ▶ sont des caractères génériques permettant de désigner un ensemble d'objets,
- ▶ s'appliquent aux arguments des commandes qui désignent des noms de fichiers
- ▶ permettent de modifier l'interprétation d'une commande.

Exemples :

- ▶ on est à la recherche d'un fichier qui commence par un "f"
- ▶ on veut lancer plusieurs commandes à la suite

LES MÉTA-CARACTÈRES

- ▶ ***** : toute chaîne de caractères (même vide)
- ▶ **?** : un unique caractère quelconque
- ▶ **[...]** : un seul caractère parmi ceux listés entre crochets
- ▶ **[!...]** : pour exclure une liste de caractères
- ▶ **;** : sépare deux commandes sur une même ligne
- ▶ **'** : délimite une chaîne de caractères contenant des espaces (à l'intérieur, tous les métacaractères perdent leur signification)
- ▶ **"** : délimite une chaîne de caractères contenant des espaces (à l'intérieur, tous les métacaractères perdent leur signification, à l'exception des métacaractères ``` et `$`) ;
- ▶ **`** : "capture" la sortie standard pour former un nouvel argument ou une nouvelle commande

UNE COMMANDE IMPORTANTE

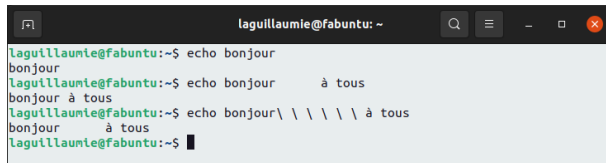
Syntaxe

`echo [SHORT-OPTION] ... [STRING] ...`

Description

Affiche sur la sortie standard l'expression après interprétation.

Exemple



```
languillaumie@fabuntu: ~  
languillaumie@fabuntu:~$ echo bonjour  
bonjour  
languillaumie@fabuntu:~$ echo bonjour      à tous  
bonjour à tous  
languillaumie@fabuntu:~$ echo bonjour\ \ \ \ à tous  
bonjour      à tous  
languillaumie@fabuntu:~$
```

LES VARIABLES

Variables simples

- ▶ Le shell permet l'utilisation des variables
- ▶ Par défaut, elles sont des chaînes de caractères
- ▶ Pour définir une variable :

```
login@machine:~$ salut=Bonjour
```

Pas d'espace autour du "=" !

- ▶ Pour accéder au contenu d'un variable, on fait précéder son nom du caractère \$
- ▶ Pour afficher une variable :

```
login@machine:~$ echo $salut
```

LES VARIABLES

- ▶ Lecture de la valeur d'une variable au clavier

```
login@machine:~$ read reponse
```

- ▶ Substitution :

```
login@machine:~$ Salut="$salut à tous"
```

- ▶ Réutilisation :

```
login@machine:~$ salut = "$salut et au revoir"
```

- ▶ Portée d'une variable : uniquement dans le shell qui l'a défini !

LES VARIABLES

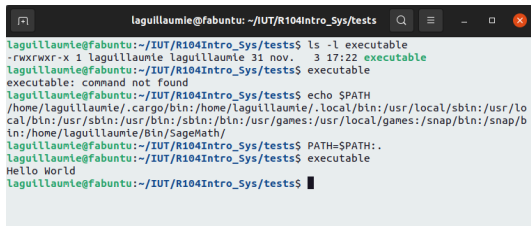
Variables d'environnement

- ▶ Pour que les variables soient connues dans les shells appelés depuis le shell, on peut les exporter
login@machine:~\$ export Salut
- ▶ Convention : on utilise des noms tout en majuscules pour les variables d'environnement
- ▶ Il existe des variables d'environnement dès le démarrage de la session
- ▶ Exemples :
 - ▶ HOME
 - ▶ PATH
 - ▶ PS1
 - ▶ USER, ...

LES VARIABLES

Lancer les exécutables :

- ▶ Supposons, que nous avons un fichier exécutable
- ▶ Il a le droit x pour l'utilisateur (rwx ---)
- ▶ En tapant son nom, il ne s'exécute pas ...
- ▶ Pour résoudre le problème :



```
laguillaumie@fabuntu: ~/IUT/R104Intro_Sys/tests
laguillaumie@fabuntu:~/IUT/R104Intro_Sys/tests$ ls -l executable
-rwxrwxr-x 1 laguillaumie laguillaumie 31 nov.  3 17:22 executable
laguillaumie@fabuntu:~/IUT/R104Intro_Sys/tests$ executable
executable: command not found
laguillaumie@fabuntu:~/IUT/R104Intro_Sys/tests$ echo $PATH
/home/laguillaumie/.cargo/bin:/home/laguillaumie/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin:/home/laguillaumie/Bin/SageMath/
laguillaumie@fabuntu:~/IUT/R104Intro_Sys/tests$ PATH=$PATH:.
laguillaumie@fabuntu:~/IUT/R104Intro_Sys/tests$ executable
Hello World
laguillaumie@fabuntu:~/IUT/R104Intro_Sys/tests$
```

Pour que la modification de la variable PATH soit toujours exécutée quand on lance un shell : il faut la mettre dans le fichier caché **.bashrc**

LES VARIABLES

Tableaux

- Création d'un tableau :

```
login@machine:~$ tab=( "un" "deux" "trois" "quatre" )
```

```
login@machine:~$ tabass=( ['un']="un" ['deux']="deux" ['trois']="trois" )
```

Possibilité de déclaration : `declare -a` (indiqué) et `declare -A` (associatif)

- Affichage du tableau :

```
login@machine:~$ echo ${tab[*]}
```

```
login@machine:~$ echo ${tab[@]}
```

- Lecture d'un élément :

```
login@machine:~$ echo ${tab[1]}
```

```
login@machine:~$ echo ${tabass["deux"]}
```

Tableaux (suite)

- Affichage de la liste des clés

```
login@machine:~$ echo ${!tabass[@]}
```

- Taille d'un tableau



```
login@machine:~$ echo ${#tabass[@]}
```




- Suppression d'un élément du tableau

```
login@machine:~$ unset tab[1]
```

EXÉCUTION DES COMMANDES

- ▶ Le shell interprète les commandes les unes après les autres
- ▶ Il lit la commande entrée par l'utilisateur, l'analyse, la prétraite et si elle est syntaxiquement correcte, l'exécute.
- ▶ L'utilisateur doit donc attendre la fin de l'exécution de la commande précédente pour que la commande suivante puisse être exécutée : on dit que l'exécution est *synchrone*.

login@machine:~\$ sleep 5  date 

- ▶ Pour terminer l'exécution d'une commande lancée en mode synchrone :  + 
- ▶ Déjà vu : plusieurs commandes séparées par des  sur une même ligne

ENTRÉE ET SORTIE STANDARDS

Un programme :

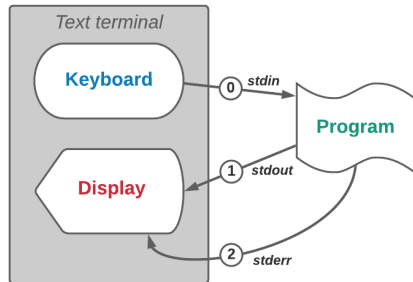
- ▶ prend des données en entrée : elles peuvent être lues dans un fichier, ou provenir d'un flux du système
- ▶ traite ces données.
- ▶ fournit en sortie des données (résultat). Elles peuvent être écrites dans un fichier, ou exportées vers un flux du système.

Trois types de **flux de données** :

- ▶ **STDIN** : entrée standard (là où sont lues les données) $\rightsquigarrow 0$
- ▶ **STDOUT** : sortie standard (là où sont écrits les résultats) $\rightsquigarrow 1$
- ▶ **STDERR** : sortie d'erreur (là où sont écrits les messages d'erreurs) $\rightsquigarrow 2$

ENTRÉE ET SORTIE STANDARDS

- ▶ Commandes lisant sur l'entrée standard
 - ▶ cat, head, tail, cut, grep...
 - ▶ elles traitent des fichiers (chemin en paramètre), ou attendent les données de l'entrée standard
 - ▶ Par défaut, l'entrée standard est le clavier
- ▶ Commandes écrivant sur la sortie standard
 - ▶ L'affichage produit par une commande est le résultat de son évaluation : celui-ci est écrit sur la sortie standard.
 - ▶ Par défaut, la sortie standard est l'écran



ENTRÉE/SORTIE : REDIRECTIONS

On peut donner une entrée et/ou une sortie alternatives à une commande :

- ▶ **login@machine:~\$ ls -a > sortie.out**
 - ▶ Redirige la **sortie standard** vers **le fichier** sortie.out
 - ▶ Si sortie.out n'existe pas, il est créé
 - ▶ Si sortie.out existe, **son contenu est écrasé** et remplacé
- ▶ **login@machine:~\$ ls -a >> sortie.out**
 - ▶ Redirige la **sortie standard** vers **le fichier** sortie.out
 - ▶ Si sortie.out n'existe pas, il est créé
 - ▶ Si sortie.out existe, l'écriture se fait **à la fin du contenu du fichier**
- ▶ **login@machine:~\$ find / -name "pass" 2> sortie.err**
 - ▶ Redirige la **sortie d'erreur** vers **le fichier** sortie.err qui sera **écrasé** s'il existe déjà
- ▶ **login@machine:~\$ find / -name "pass" 2>> sortie.err**
 - ▶ Redirige la **sortie d'erreur** vers **le fichier** sortie.err avec **écriture en fin de fichier**

ENTRÉE/SORTIE : REDIRECTIONS

On peut donner une entrée et/ou une sortie alternatives à une commande :

- ▶ **login@machine:~\$ wc fichier.in** VS. **login@machine:~\$ wc < fichier.in**
 - ▶ Dans le premier cas, wc sait qu'il lit son entrée dans `fichier.in`.
 - ▶ Dans le second, wc sait qu'il lit depuis son entrée standard (donc ne connaît pas le nom du fichier).

- ▶ **login@machine:~\$ cat > fichier.out**
toto ctrl + D

- ▶ Pour oublier la sortie :
 - ▶ **login@machine:~\$ ls -a > /dev/null**
 - ▶ **login@machine:~\$ find / -name "pass" 2> /dev/null**

ENTRÉE/SORTIE : REDIRECTIONS

- Pour envoyer la sortie standard et la sortie d'erreur dans un fichier :


```
login@machine:~$ prog > file 2>&1
```


En fait, renvoie la sortie d'erreur dans la sortie standard.


- Pour lire au clavier jusqu'à "c" en début de ligne :

```
login@machine:~$ wc << c
```

```
login@machine:~$ cat << toto >/tmp/test
```

```
> bla 
```

```
> bli 
```

```
> toto 
```

ENTRÉE/SORTIE : REDIRECTIONS

Redirections vers d'autres terminaux

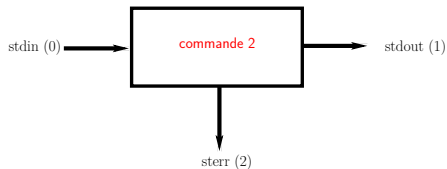
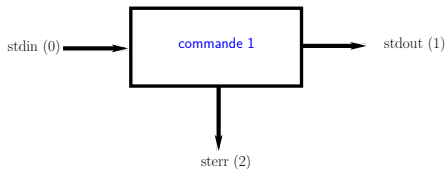
- ▶ Chaque terminal correspond à un fichier dans le système de fichiers
- ▶ Les terminaux (comme le matériel) se trouve dans /dev
- ▶ **login@machine:~\$ tty**

```
laguillaumie@fabuntu: ~  
laguillaumie@fabuntu:~$ tty  
/dev/pts/4  
laguillaumie@fabuntu:~$ rm toto.txt  
rm: cannot remove 'toto.txt': No such file or directory  
laguillaumie@fabuntu:~$ rm toto.txt 2>/dev/pts/5  
laguillaumie@fabuntu:~$
```

```
laguillaumie@fabuntu: ~  
laguillaumie@fabuntu:~$ tty  
/dev/pts/5  
laguillaumie@fabuntu:~$ rm: cannot remove 'toto.txt': No such file or directory  
laguillaumie@fabuntu:~$
```

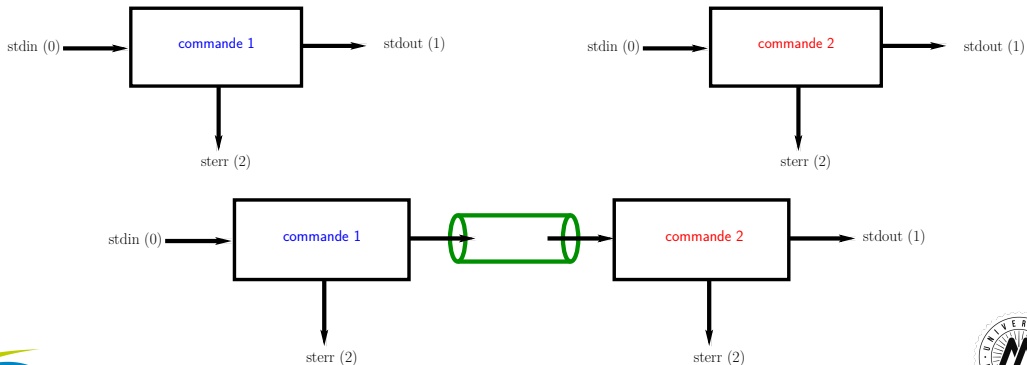
TUBES (PIPE)

Un tube permet de rediriger la sortie standard d'une commande vers l'entrée standard d'une autre commande.



TUBES (PIPE)

Un tube permet de rediriger la sortie standard d'une commande vers l'entrée standard d'une autre commande.



TUBES (PIPE)

Un tube permet de rediriger la sortie standard d'une commande vers l'entrée standard d'une autre commande.

- ▶ `commande1` | `commande2`
- ▶ Un fichier particulier (un « tube », un « pipe ») est utilisé entre les deux processus
 - ▶ `commande1` écrit dans le tube
 - ▶ `commande2` peut lire du tube ce que `commande1` a écrit

TUBES (PIPE)

Un tube permet de rediriger la sortie standard d'une commande vers l'entrée standard d'une autre commande.

▶ `login@machine:~$ ls | more`

▶ `login@machine:~$ ls /etc | grep conf`

▶ `login@machine:~$ ls /etc | grep conf | more`

▶ `login@machine:~$ ls /bin /usr/local/bin | grep "cp" | wc -l`

Schéma général

- ▶ Le premier mot est considéré comme le nom d'un exécutable qui peut être suivi par des options et des arguments
- ▶ Le shell :
 - ▶ découpe la commande en jetons
 - ▶ vérifie les alias
 - ▶ vérifie les *built-ins*
 - ▶ trouve la commande dans le PATH
 - ▶ lance le programme
 - ▶ quand il est fini, affiche le prompt défini par la variable PS1
 - ▶ attend la nouvelle commande.

Exécution séquentielle, parallèle et conditionnelle

- ▶ Exécution séquentielle (par défaut)
 - ▶ Plusieurs commandes sur la même ligne doivent être séparées par des ;
 - ▶ Exemple : **login@machine:~\$** cd ; pwd ; ls
- ▶ Exécution parallèle
 - ▶ Des composants des tubes sont lancés parallèlement
- ▶ Exécution conditionnelle
 - ▶ `cmd1 && cmd2` → `cmd2` est exécutée ssi `cmd1` retourne 0
 - ▶ `cmd1 || cmd2` → `cmd2` est exécutée ssi `cmd1` retourne une valeur différente de 0

Substitutions

- ▶ En général, les substitutions (\$var) sont réalisées *avant* l'exécution
- ▶ Cas particuliers
 - ▶ Chaînes des caractères protégées par ' '
 - Exemple : `echo '$PATH'`
 - ▶ Chaînes des caractères délimitées par " "
 - Exemple : `echo "$PATH"`
 - ▶ Chaînes des caractères entre ` ` (accents graves)
 - Exemple : `echo `wc text.txt``

```
laguillaunie@fabuntu: ~  
laguillaunie@fabuntu:~$ echo $PATH  
/home/laguillaunie/.cargo/bin:/home/laguillaunie/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin:/home/laguillaunie/Bin/SageMath  
laguillaunie@fabuntu:~$ echo " $PATH "  
/home/laguillaunie/.cargo/bin:/home/laguillaunie/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin:/home/laguillaunie/Bin/SageMath/  
laguillaunie@fabuntu:~$ echo ` $PATH `  
$PATH  
laguillaunie@fabuntu:~$ echo ` $PATH `  
bash: /home/laguillaunie/.cargo/bin:/home/laguillaunie/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin:/home/laguillaunie/Bin/SageMath/:  
No such file or directory  
laguillaunie@fabuntu:~$ echo `wc /lib/R/library/Matrix/include/cho1nod.h`  
1064 7396 49144 /lib/R/library/Matrix/include/cho1nod.h  
laguillaunie@fabuntu:~$
```

LA COMMANDE FIND

Syntaxe

```
find [-H] [-L] [-P] [-D debugopts] [-Olevel] [starting-point...]
      [expression]
```

Description

Cherche un fichier dans une hierarchie de répertoires

- recherche multi-critères

Exemples

- `login@machine:~$ find /etc -name password`
- `login@machine:~$ find /etc -name 'p*d' -print 2>/dev/null`
- `login@machine:~$ find /etc -name 'p*d' -type d -print 2>/dev/null`
- `login@machine:~$ find /etc -name passwd -name shadow -print 2>/dev/null`
- `login@machine:~$ find /etc /usr -name passwd -print 2>/dev/null`
- `login@machine:~$ find /usr -size +10M -print -exec ls -lh {} \; 2>/dev/null`

