

Initiation à Unix

Cours 1 - Présentation

Université Henri Poincaré
Master Génomique et Informatique

Automne 2008

Plan

Introduction

L'interpréteur de commande

Le système de fichiers

Les commandes fondamentales

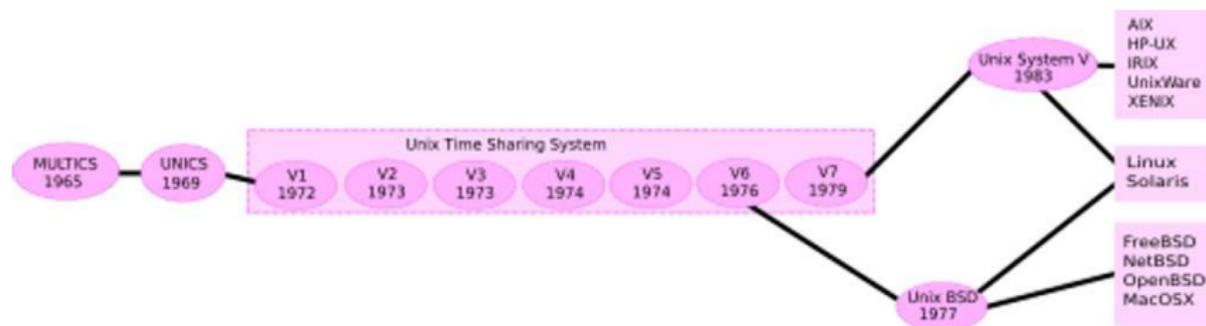
Les commandes d'administration

Les variables d'environnement

Historique

- ▶ 1965: *Multics* (laboratoires Bell - AT&T, MIT, General Electric)
- ▶ 1969: *Unics* (Ken Thompson, laboratoires Bell, développé en langage d'assemblage)
- ▶ 1971: publication de *The UNIX Programmer's manual*
- ▶ 1973: réécriture de *Unix* en langage C (Dennis Ritchie, Brian Kernighan)
- ▶ fin des années 70: reprise par le monde académique (Université de Californie à Berkeley)

Historique (suite)



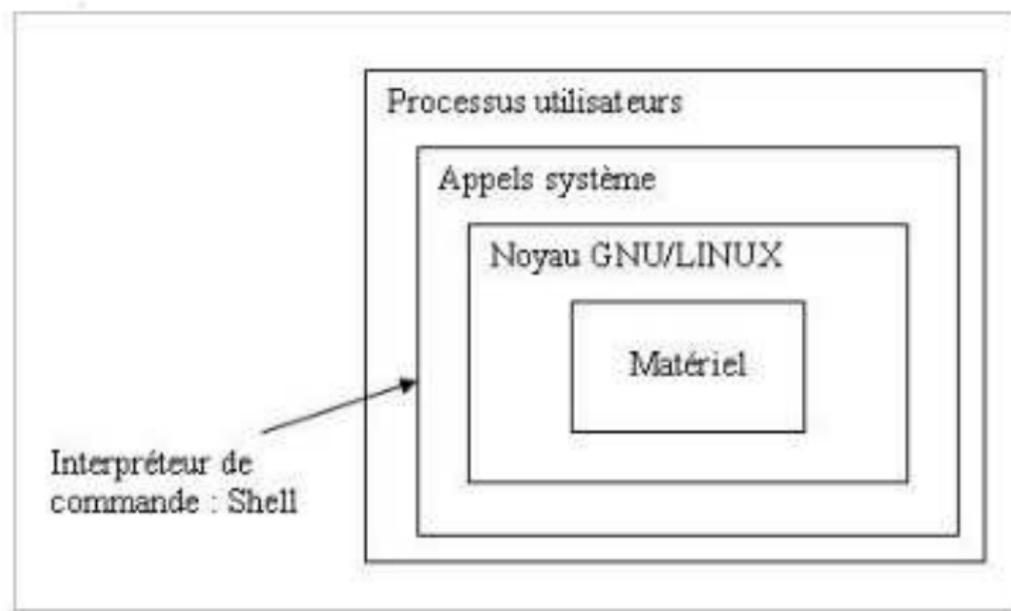
(source: Wikipedia)

Définition

UNIX est un système d'exploitation permettant de contrôler un PC et ses différents périphériques. UNIX se distingue par les caractéristiques suivantes :

- ▶ **multi-utilisateurs** (qui peut être utilisé simultanément par plusieurs personnes)
- ▶ **multi-tâches** (un utilisateur peut exécuter plusieurs programmes en même temps)
- ▶ repose sur un **noyau** (kernel) utilisant 4 concepts principaux **fichiers, droits d'accès, processus** et **communication interprocessus** (IPC)

Schéma d'UNIX



Plan

Introduction

L'interpréteur de commande

Le système de fichiers

Les commandes fondamentales

Les commandes d'administration

Les variables d'environnement

L'interpréteur de commande

- ▶ **Shell** : interface entre l'utilisateur et le système d'exploitation ("coquille")
- ▶ Application (fichier exécutable) chargé d'interpréter les commandes des utilisateurs et de les transmettre au système
- ▶ Différents types de shell, les principaux étant :
 - **sh** (Bourne shell)
 - **bash** (Bourne again shell)
 - **csh** (C shell)
 - **Tcsh** (Tenex C shell)
 - **ksh** Korn shell
 - **zsh** Zero shell
- ▶ Le nom du shell correspond généralement au nom de l'exécutable :
% /bin/bash

Utilisation du shell

- ▶ Le shell correspond à une fenêtre présentant un *prompt*, encore appelé *invite de commande*. Celle-ci est paramétrable et par défaut en bash se compose comme suit :

```
login@machine$
```

(suffixe \$ → utilisateur normal,

suffixe # → super-utilisateur – administrateur)

- ▶ On saisit les commandes à la suite du prompt
- ▶ Pour stopper la commande en cours: Ctrl-C
- ▶ Pour mettre en attente la commande en cours: Ctrl-Z
- ▶ Pour terminer l'entrée standard (les éventuelles paramètres donnés par l'utilisateur via le clavier): Ctrl-D

Utilisation du shell (suite)

- ▶ Le shell est personnalisable au moyen des fichiers suivants :
 - 1 le fichier `/etc/profile`, s'il existe
 - 2 le fichier `$HOME/.bash_profile`, s'il existe
 - 3 le fichier `$HOME/.bash_login`, s'il existe
 - 4 le fichier `$HOME/.profile`, s'il existe
 - 5 le fichier système `/etc/bashrc`
 - 6 le fichier caché `.bashrc`, s'il existe

Les entrées-sorties standards

- ▶ Lors de l'exécution d'une commande, un processus est créé. Celui-ci va alors ouvrir trois flux :
 - `stdin` l'**entrée standard**, par défaut le clavier, identifiée par l'entier **0** (descripteur)
 - `stdout` la **sortie standard**, par défaut l'écran, identifiée par l'entier **1**
 - `stderr` la **sortie d'erreur standard**, par défaut l'écran, identifiée par l'entier **2**

Les redirections

Il est possible de redigirer les flux d'entée-sortie au moyen d'opérateurs spécifiques :

- > redirection de la sortie standard (par exemple dans un fichier)
- < redirection de l'entrée standard
- >> redirection de la sortie standard avec **concaténation**
- > & redirection des sorties standard et d'erreur
- >! redirection avec écrasement de fichier
- | redirection de la sortie standard vers l'entrée standard (pipe)

Exemple: la commande echo

```
$ echo "ca va"
```

```
ca va
```

```
$ java toto
```

```
Exception in thread "main"
```

```
java.lang.NoClassDefFoundError: toto
```

```
$ java toto > erreur.txt
```

```
Exception in thread "main"
```

```
java.lang.NoClassDefFoundError: toto
```

```
$ java toto > & erreur.txt
```

Plan

Introduction

L'interpréteur de commande

Le système de fichiers

Les commandes fondamentales

Les commandes d'administration

Les variables d'environnement

Le système de fichiers

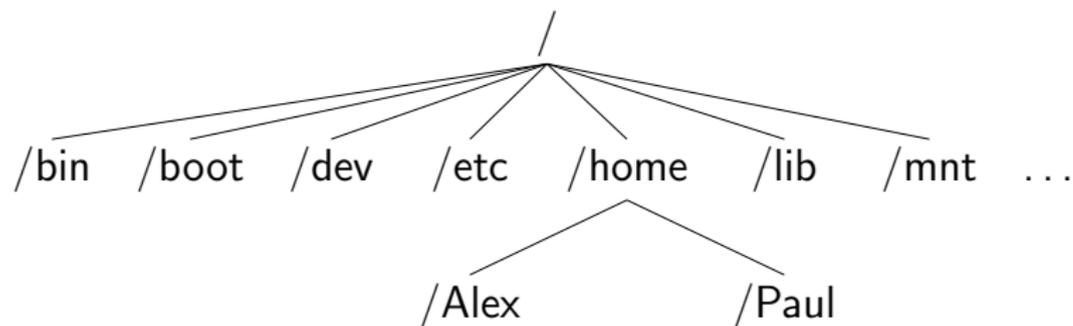
- ▶ Le système de fichier correspond à une arborescence que l'on parcourt de la racine (root) vers les feuilles
- ▶ La racine se note / (slash)
- ▶ Il s'agit d'un répertoire contenant les sous-répertoires suivants :
 - /bin exécutables essentiels pour le système, directement utilisable par les utilisateurs
 - /boot contient les fichiers permettant à Linux de démarrer
 - /dev contient les points d'entrée des périphériques (=device)
 - /etc configuration du réseau
→ contient les commandes et les fichiers nécessaires à l'administrateur du système (fichiers passwd, group, inittab, ld.so.conf, lilo.conf, ...)

Le système de fichiers (suite)

► Sous-répertoires de la racine (suite) :

- /home répertoire personnel des utilisateurs
- /lib contient des bibliothèques partagées essentielles au système lors du démarrage
- /mnt contient les points de montage des partitions temporaires (cd-rom, disquette, ...), parfois nommé *media*
- /opt contient des packages d'applications supplémentaires
- /proc fichiers content des info sur la mémoire, E/S, périphérique, compatibilité pour le noyau, ...
- /root répertoire de l'administrateur root
- /usr hiérarchie secondaire (utilisateurs)
- /var contient des données variables
- /tmp contient les fichiers temporaires

Représentation graphique



Plan

Introduction

L'interpréteur de commande

Le système de fichiers

Les commandes fondamentales

Les commandes d'administration

Les variables d'environnement

Les commandes fondamentales

- ▶ Aide

\$ man commande

→ Manuel pour les commandes

- ▶ Où suis-je dans l'arborescence ?

\$ pwd

NB: chemin absolu vs chemin relatif

Exemple:

```
yannick@nausicaa:~/toto $ pwd  
/home/yannick/toto
```

Les commandes fondamentales (suite)

- ▶ Comment se déplacer dans l'arborescence ?

`cd [chemin]`

→ Permet de changer de répertoire (**c**hange **d**irectory)

Alias :

`.` → répertoire courant

`..` → répertoire parent

Exemples :

`$ pwd → /home/yannick/toto`

`$ cd .. → /home/yannick/`

`$ cd projet → /home/yannick/projet`

`$ cd /usr/local → /usr/local`

Les commandes fondamentales (suite)

- Lister le contenu d'un répertoire ?

```
ls [option] [chemin]
```

→ Liste le contenu d'un répertoire avec plus ou moins de détails

Exemples :

\$ `ls l*` → liste tous les fichiers commençant par l

\$ `ls -l` → liste tous les fichiers du répertoire courant, en donnant les attributs des fichiers (droits, taille, etc)

\$ `ls -a` → liste tous les fichiers du répertoire courant (y compris les fichiers cachés dont le nom commence par un ".")

\$ `man ls` → affiche la page de manuel de la commande ls

Les commandes fondamentales (suite)

- ▶ Visualiser le contenu d'un fichier ?
`cat [option] [chemin vers le fichier1, fichier 2, etc]`
→ affiche le contenu d'un fichier

Exemples:

`$ cat .bash_profile` → affiche le contenu du fichier
caché `.bash_profile`

`$ cat toto > tata` → écrit le contenu du fichier `toto`
dans un fichier nommé `tata`

- ▶ Visualiser le contenu d'un fichier page à page ?
`more [fichier]`
- ▶ Visualiser le contenu d'un fichier dans un flux ?
`less [fichier]`

Les commandes fondamentales (suite)

- ▶ Obtenir des statistiques sur le contenu d'un fichier ?
`wc [option] [chemin vers le fichier]`
 → affiche le nombre de mots / lignes / caractères d'un fichier

Exemples :

`$ wc -l toto` → affiche le nombre de lignes du fichier toto

`$ wc -c toto` → affiche le nombre de caractères du fichier toto

`$ ls | wc -l` → affiche le nombre de fichiers dans le répertoire courant

- ▶ Editer un fichier ?
`emacs [fichier]`
`vim [fichier]`
`gedit [fichier]`
 ...

Les commandes fondamentales (suite)

► Copier un fichier ?

```
cp [option] [chemin vers fichier source]  
[chemin vers fichier destination]
```

→ copie un fichier source en le renommant si le chemin du fichier destination contient un nom de fichier

Exemples :

```
$ cp toto /tmp/ → copie le fichier local toto dans /tmp  
(toujours nommé toto)
```

```
$ cp toto /tmp/tata → copie le fichier local toto dans  
/tmp en le nommant tata
```

```
$ cp -r projet /tmp → copie le contenu du répertoire  
projet dans le répertoire /tmp/projet
```

Les commandes fondamentales (suite)

- ▶ Déplacer un fichier ?

```
mv [option] [chemin vers fichier source]  
[chemin vers fichier destination]
```

→ déplace un fichier source en le renommant si le chemin du fichier destination contient un nom de fichier

Exemples :

```
$ mv toto /tmp/ → déplace le fichier local toto dans  
/tmp (toujours nommé toto)
```

```
$ mv toto /tmp/tata → déplace le fichier local toto  
dans /tmp en le nommant tata
```

```
$ mv -i toto /tmp → déplace le fichier toto dans /tmp  
en prévenant l'utilisateur s'il existe déjà un fichier  
/tmp/toto
```

Les commandes fondamentales (suite)

- Supprimer un fichier ?

```
rm [option] [chemin vers fichier]
```

→ supprime un fichier

Exemples :

```
$ rm toto → supprime le fichier toto
```

```
$ rm -i toto → supprime le fichier toto en demandant confirmation à l'utilisateur
```

```
$ rm -f toto* → supprime les fichiers dont le nom commence par toto, sans demander confirmation à l'utilisateur
```

```
$ rm -r projet → efface récursivement le contenu du répertoire projet
```

Les commandes fondamentales (suite)

- ▶ Créer / supprimer un répertoire ?
mkdir [chemin vers répertoire]
rmdir [chemin vers répertoire]
→ crée / supprime un répertoire *vide*

Exemples:

```
$ mkdir toto → crée le répertoire toto
```

```
$ rmdir toto → supprime le répertoire vide toto
```

```
$ rmdir projet → rmdir: projet/: Directory  
not empty
```

Les commandes fondamentales (suite)

- ▶ Retrouver un fichier ?

```
find [options]
```

→ effectue une recherche à partir des informations données en option

Exemples:

```
$ find . -name toto → cherche, dans le répertoire courant et ses sous-répertoires, un fichier nommé toto
```

```
$ find /tmp/ -type d → cherche tous les sous-répertoires du répertoire /tmp
```

```
$ find /tmp -type d -exec ls '{}' \; → affiche le contenu des sous-répertoires du répertoire /tmp
```

Les commandes fondamentales (suite)

- ▶ Retrouver un fichier exécutable ?

`which` commande

→ effectue une recherche dans la liste des exécutables de la commande donnée

Exemples:

`$ which echo` → `/bin/echo`

- ▶ Connaître l'espace occupé par un répertoire / disque ?

`du [option] fichier`

→ donne la taille en octets d'un fichier

`df [option]`

→ donne la taille des données présentes sur chaque disque

Exemples:

`du -sh projet` → `4.0K projet/`

Les commandes fondamentales (suite)

- Rechercher un motif dans un fichier ?

```
grep [options] expression régulière fichier1
...
```

→ effectue une recherche à partir d'un motif fourni dans une expression régulière donnée

Exemples:

\$ grep "listeria" /home/Cath/cours/* → cherche, dans les fichiers du répertoire cours, des fichiers contenant le motif listeria

\$ grep -n "listeria" /home/Cath/cours/* → idem, mais en affichant le numéro de ligne

\$ grep -c "listeria" /home/Cath/cours/* → idem, mais en donnant le nombre d'occurrences du motif

Les commandes fondamentales (suite)

- ▶ Compresser/décompresser un fichier ?

`gzip fichier`

→ compresse un fichier au format `.gz` (algorithme *deflate*)

`gunzip fichier`

→ décompresse un fichier au format `.gz`

Exemples:

```
$ gzip toto.txt → toto.txt.gz
```

```
$ gunzip toto.txt.gz → toto.txt
```

Les commandes fondamentales (suite)

- ▶ Créer/extraire une archive ?

```
tar cf projet.tar projet/*
```

→ crée une archive contenant le contenu du répertoire projet et nommée projet.tar

```
tar xf projet.tar
```

→ extrait le contenu de l'archive nommée projet.tar

```
tar zcf projet.tar projet/*
```

→ crée et compresse une archive contenant le contenu du répertoire projet et nommée projet.tar.gz

```
tar zxf projet.tar.gz
```

→ extrait le contenu de l'archive compressée nommée projet.tar.gz

Les commandes fondamentales (suite)

- ▶ Créer un lien vers un fichier ?

```
ln [options] fichier_source lien
```

→ création d'un lien. un lien est un type spécial de fichier qui permet à plusieurs noms de fichiers de faire référence au même fichier sur le disque.

NB: lien "dur" vs lien symbolique

Exemples:

```
$ ln /home/yannick/cours.pdf
```

/home/yannick/projet/cours.pdf → le fichier cours.pdf du répertoire yannick du répertoire projet est un lien vers le fichier cours.pdf du répertoire yannick

```
$ ln -s /home/yannick/cours.pdf
```

/home/yannick/projet/cours.pdf → idem avec un lien symbolique

NB: que fait `rm /home/yannick/projet/cours.pdf` ?



Les commandes fondamentales (suite)

- ▶ Connaître les ressources prises par une commande ?

`ps [options]`

→ donne des informations sur les processus en cours

Exemple :

`$ ps ux →`

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	%COMMAND
yannick	6316	0.0	0.0	13272	1728	?	SL	09:26	0:00	/bin/echo

- ▶ Connaître l'activité du système ?

`$ top`

→ donne des informations sur l'activité du système
(ressources occupées, etc)

Les commandes fondamentales (suite)

- ▶ Interrompre un processus ?
\$ kill [option] PID
→ envoie un signal au processus identifié par le nombre PID
(si l'option est -9, ce signal provoque l'interruption de la commande)
- ▶ Trier le contenu d'un fichier ?
\$ sort fichier
- ▶ Savoir qui est connecté au système ?
\$ who

Plan

Introduction

L'interpréteur de commande

Le système de fichiers

Les commandes fondamentales

Les commandes d'administration

Les variables d'environnement

Les commandes d'administration

- Gérer les droits d'un fichier ?

```
chmod [options] droits fichier1, fichier2, ...
```

→ change les droits d'un fichier

Les droits sont définis comme suit :

- u droits de l'utilisateur (user)
- g droits des utilisateurs du groupe (group)
- a droits de tous les utilisateurs (all)
- +r droit en lecture accordé
- r droit en lecture retiré
- +w droit en écriture accordé
- +x droit d'exécution accordé

Exemples :

```
$ chmod a+r toto.txt → autorise l'accès en lecture du  
fichier toto.txt à tout le monde
```

```
$ chmod 444 toto.txt → idem
```



Les commandes d'administration (suite)

- ▶ Changer le propriétaire et groupe d'un fichier ?
`chown [options] utilisateur.groupe fichier`
→ change le propriétaire d'un fichier
NB: nécessite d'être administrateur (super-user)

Exemple :

```
$ sudo chown -R paul.L1 projet/ → définit  
l'utilisateur paul et le groupe L1 au répertoire projet et à  
tous ses fichiers
```

- ▶ Ajouter un utilisateur ?
`useradd [options] login`
- ▶ Changer de mot de passe ?
`passwd [options] login`

Plan

Introduction

L'interpréteur de commande

Le système de fichiers

Les commandes fondamentales

Les commandes d'administration

Les variables d'environnement

Les variables d'environnement

- ▶ Variables permettant de paramétrer le fonctionnement du système (langue utilisée, chemins vers les fichiers exécutables, chemin vers les librairies, etc)
- ▶ Variables principales :
 - SHELL interpréteur de commande utilisé
 - HOME chemin du répertoire de l'utilisateur
 - PATH chemin des exécutables
- ▶ Lire une variable d'environnement : `echo $HOME`
- ▶ Définir une variable d'environnement (bash) :
`export PATH=$PATH:/home/yannick/myexec`
- ▶ De nombreuses variables d'environnement sont définies dans le fichier `$HOME/.bash_profile` (bash)

Ressources en ligne

- ▶ les pages man !
- ▶ <http://www.ee.surrey.ac.uk/Teaching/Unix/>
- ▶ <http://www.lmd.ens.fr/Ressources-Info/Unix-Doc/html/cours-unix.html>
- ▶ <http://www.commentcamarche.net/unix/unixcomm.php3>
- ▶ Remerciements: ce cours a été réalisé à partir du support de Catherine Eng.