LE SYSTÈME UNIX

Guillaume Chanel

Remerciements à Jean-Luc Falcone

Septembre 2019

SHELL

INTERFACE TEXTE

Pourquoi utiliser une interface texte en 2017 ?

LE SHELL

- Interface textuelle, ligne de commande
- presque toutes les fonctionnalités d'Unix sont accessibles
- Petits outils simple que l'on peut composer
- Possibilité d'écrire des **scripts**

LES SHELLS

- Il existe plusieurs shell différents (sh, **bash**, csh, zsh, ash, etc.)
- Ils diffèrent par:
  - Manière d'écrire les scripts
  - Configuration et personnalisation
  - Commandes prédéfinies

TERMINAL & PSEUDO-TERMINAL

terminal

TERMINAL & PSEUDO-TERMINAL

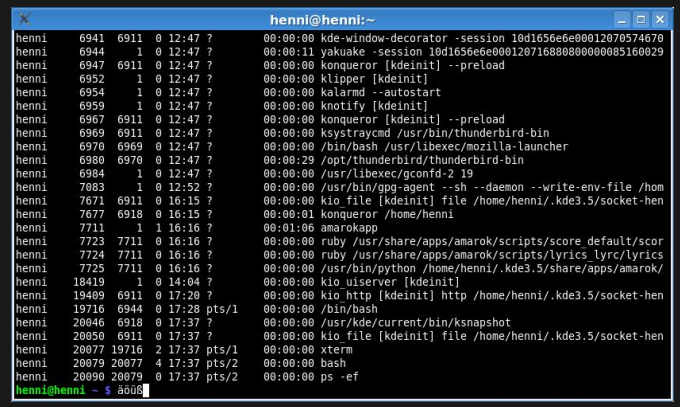
terminal

VT220

TERMINAL & PSEUDO-TERMINAL

terminal

VT220



Pseudo-terminal

UTILISATEURS

UTILISATEURS

- Chaque utilisateur se loge sur une machine Unix avec une identité (*user*)
- Un utilisateur est décrit par:
  - Nom d'utilisateur
  - Mot de passe
  - Identifiant numérique (*userid*)
  - Groupe par défaut
  - Répertoire `home`
  - Description
  - Shell par défaut

Notes

- On peut créer un utilisateur pour certain programmes (par exemple serveur web).
- Le utilisateurs "locaux" sont listés dans le fichier `/etc/passwd`.

PROPRIÉTÉ & PERMISSIONS

- Chaque processus appartient à un utilisateur
- Chaque fichier et répertoire appartient à un utilisateur
- Un utilisateur est abilité à:
  - Modifier ses fichiers et ses répertoires
  - Intéragir (et arrêter) ses processus

SUPER-UTILISATEUR (ROOT)

- Il existe un super-utilisateur: **root**
- Son userid est le 0
- Il a le droit de tout faire.

Attention

Accès au mot de passe root: faille de sécurité majeure.

GROUPES

Les utilisateurs peuvent faire partie d'un ou plusieurs **groupes**:

- Groupe par défaut
- Autres groupes

Ceci permet par exemple de:

- restreindre l'accès à un répertoire à un seul groupe
- donner accès à un périphérique à un seul groupe

Note

Les groupes "locaux" sont listés dans le fichier `/etc/group`.

COMMANDES IMPORTANTES

whoami	quel est mon nom d'utilisateur
groups	de quels groupes fais-je partie ?
id	affiche les identifiants de mon nom d'utilisateur et de mes groupes
w	quels sont les utilisateurs logués sur la machine
who	comme **w** mais avec moins d'infos
exit	permet de quitter le shell courant

MANUEL

ACCÈDER AU MANUEL (`man`)

- Un manuel très complet est présent sur les systèmes Unix
- Commande:

```
man [section] <sujet>
```

Exemples

```
$ man pwd
```

```
$ man 3 getcwd
```

```
$ man man
```

SECTIONS DU MANUEL SUR GNU/LINUX

1	Commandes générales
2	Appels systèmes
3	Librairie C
4	Fichiers spéciaux
5	Formats de fichiers et conventions
6	Jeux
7	Divers
8	Commandes d'administration et démons


RTFM !
Read the fucking manual!

FICHIERS

SYSTÈME DE FICHIER (1)

- Manière d'organiser les données sur le disque:
  - Noms
  - Répertoires
  - Permissions
  - Metadonnées
  - Maintien de l'intégrité
  - …

SYSTÈMES DE FICHIER (2)

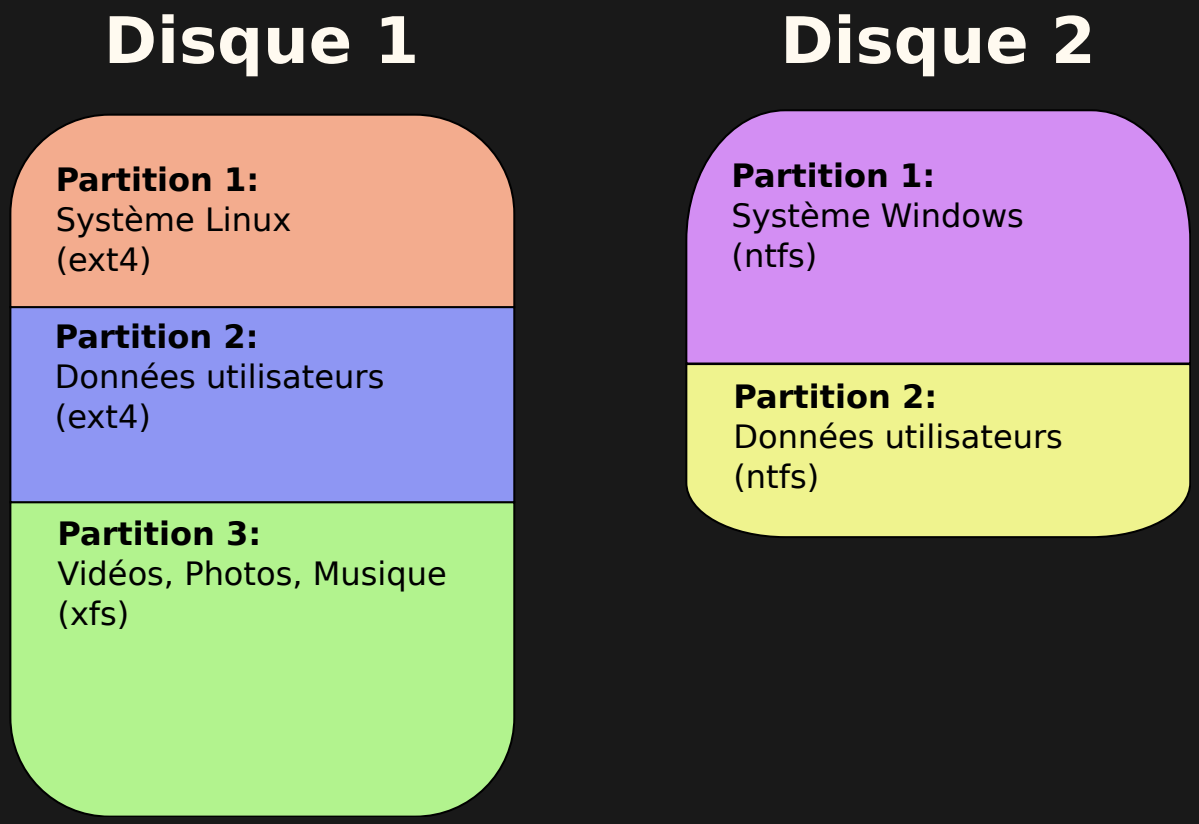
Il existe plusieurs types de systèmes de fichiers. Par exemple:

VFAT	Ancien système de fichier de Windows, très utilisé sur les clé-usb, lecteurs MP3, etc.
NTFS	Système de fichier récent sur Windows.
ext4	Système de fichier par défaut sous beaucoup de distributions GNU/Linux
ISO9660	Système de fichier des CD-ROMs
UDF	Système de fichier des DVDs
HFS+	Système de fichier sur MacOSX
NFS	Système de fichier **réseau** sous Unix

ARBORESCENCE (1)

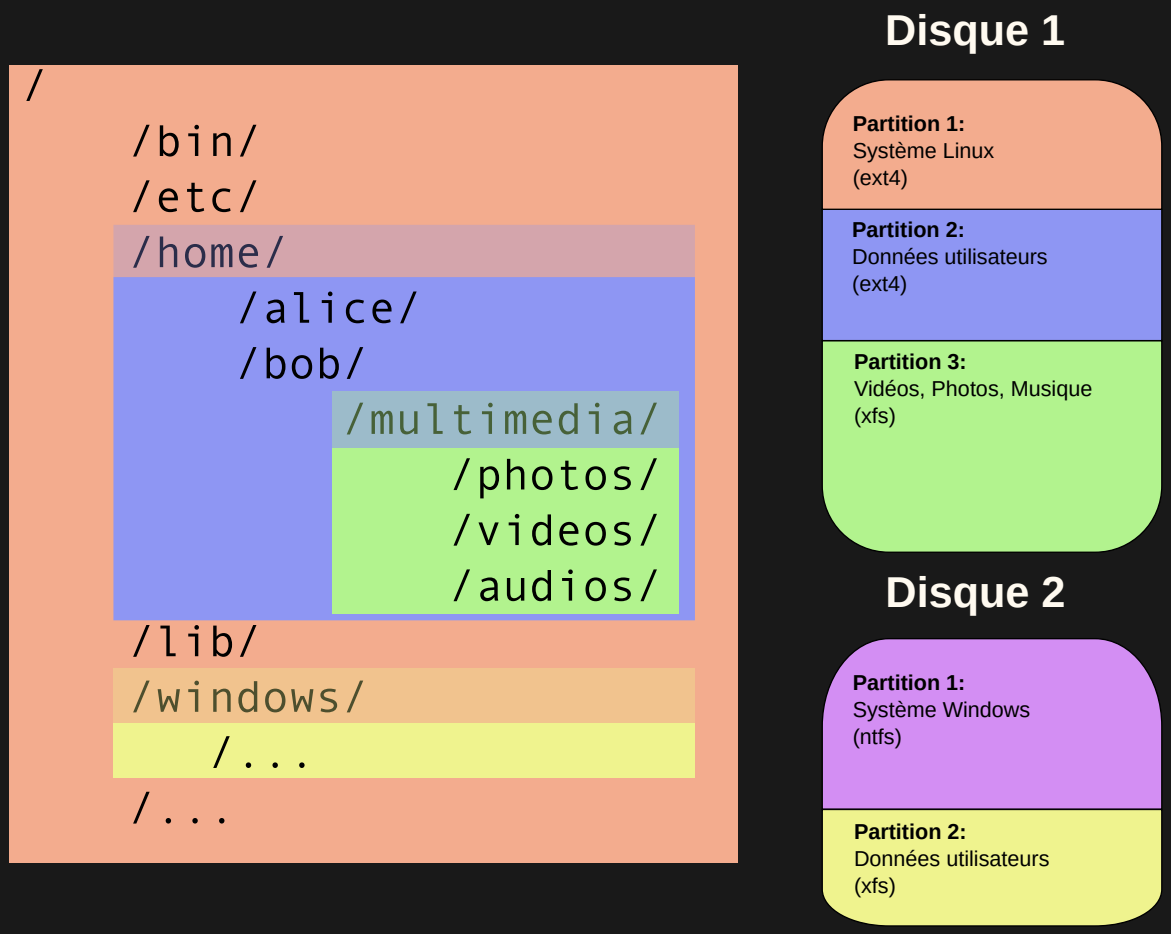
- Tous les systèmes de fichiers sont "montés" sur une seule et même arborescence.
- Seul `root` peut monter/démonter des systèmes de fichiers (par défaut)
- La plupart des installations GNU/Linux actuelles, montent automatiquement les disques externes (CD, DVD, usb)

ARBORESCENCE (2)



Partitionnement des espaces

ARBORESCENCE (3)



Montage des partitions dans l'arborescence

ARBORESCENCE UNIX STANDARD

`/bin/`	Principaux exécutables
`/boot/`	Fichiers de démarrage
`/dev/`	*Périphériques*
`/etc`	Configuration système
`/home/`	Données utilisateurs
`/lib/`	Librairies système
`/media/`	Montage des périphériques de stockage
`/mnt/`	Point de montage manuel
`/proc/`	*Processus*
`/root/`	Répertoire perso du superutilisateur
`/sbin/`	Exécutables pour la maintenance
`/sys/`	*Informations système*
`/usr/`	Applications et librairies utilisateurs
`/var/`	Données variables (logs, spool, mail…)

NOMS DE FICHIERS ET DE RÉPERTOIRES

- Maximum 255 caractères
- Tous les caractères sont autorisés, mais déconseillés
  - espaces
  - . * ? / !
- Si le nom du fichier commence par un point, il est **caché**.

CHEMIN D'ACCÈS

Les conventions suivantes (POSIX) s'appliquent:

- La racine est représentée par /
- Les répertoires sont séparés par /
- Tout chemin qui ne commence par / est relatif au répertoire courant.
- Le répertoire courant est symbolisé par .
- Le réperoire parent est symbolisé par ..
- Le répertoire *home* est symbolisé par ~

WTF?

```
~/./././/./hello
```

```
~/./hello/../hello/../.
```

COMMANDES UTILES

pwd	affiche le répertoire courant
cd	change de répertoire courant
ls	liste le contenu d'un répertoire

LISTER UN RÉPERTOIRE (`ls`)

```
ls [options]... [répertoire-ou-fichier]...
```

Par défaut, utilise le répertoire courant

Options courantes:

- **-l** liste les fichiers (plus d'infos)
- **-h** affiche les tailles en format humain
- **-a** affiche les fichiers cachés

```
$ ls ..
$ ls dossier/sous-dossier
$ ls -l dossier1 dossier2 fichier
$ ls -l | less # utile si la sortie est trop longue
```

CRÉER UN REPERTOIRE (`mkdir`)

```
mkdir [options]... répertoire-à-créer...
```

L'option `-p` permet de créer automatiquement les repertoires parents.

```
$ mkdir ../foo
$ mkdir ../foo/bar ../foo/bar/baz

$ mkdir -p ../foo/bar/baz  # equivalent des précédents en une commande
```

DÉPLACER ET RENOMMER DES FICHIERS (mv)

```
mv [options]... fichiers... destination
```

Fonctionne aussi avec des répertoires

```
$ mv foo.avi bar.mpg baz.mkv video/  # deplace foo.avi et bar.mpg dans le dossier vidéo
$ mv SysInfo/ InfoBio ~/cours  # deplace les premiers éléments dans le répertoire cours du "home"

$ mv foo.AVI foo.avi  # renomme le fichier foo.AVI vers foo.avi (case sensitive)

$ mkdir bar
$ mv bar/ foo/  # renomme le dossier bar en foo SI foo n'existe pas !
```

OPTIONS DE LA COMMANDE `mv`

Que faire si on déplace un fichier vers une source ou un fichier existe déjà avec le même nom:

- **-f** Ecrase silencieusement la destination si elle existe (*force*)
- **-i** Demande une confirmation avant d'écraser la destination (*interactive*)
- **-u** Efface la destination si elle est plus ancienne que la source (*update*)

COPIER UN FICHIER (cp)

```
cp [options]... fichiers... destination
```

Options:

-r Copie aussi le contenu des répertoires (*recursive*)

-a Préserve les attributs (permissions, propriétaire, etc.) et copie le contenu des répertoires (*archive*)

```
$ cp foo.txt foo.txt.backup
$ cp bar.txt baz.doc projet/  # copie les deux fichiers dans le dossier projet
```

OPTIONS DE LA COMMANDE `cp`

Que faire si on déplace un fichier vers une source ou un fichier existe déjà avec le même nom:

- **-f** Ecrase silencieusement la destination si elle existe (*force*)
- **-i** Demande une confirmation avant d'écraser la destination (*interactive*)
- **-u** Efface la destination si elle est plus ancienne que la source (*update*)

EFFACER UN RÉPERTOIRE VIDE (`rmdir`)

```
rmdir [options]... répertoires...
```

L'option `-p` permet de supprimer automatiquement les repertoires parents si ils sont vides.

```
$ rmdir ../foo/bar/baz
$ rmdir ../foo/bar
$ rmdir ../foo

$ rmdir ../foo/bar/baz  # equivalent aux commandes précédentes
```

EFFACER UN FICHIER / RÉPERTOIRE (`rm`)

```
rm [options]... fichiers...
```

Options:

- **-r** efface recursivement le contenu (*recursive*)
- **-f** ne demande pas de confirmation (*force*)
- **-i** demande une confirmation pour chaque fichier (*interactive*)

```
$ rmdir ../foo/bar/baz
$ rm foo.txt foo.txt.backup
$ rm -rf foo/  # supprime tout le contenu de foo
```

Y'a-t-il une différence ?

```
$ mv foo.avi video/
```

vs.

```
$ cp foo.avi video/
$ rm foo.avi
```

A. oui

B. non

Y'a-t-il une différence ?

```
$ mv foo.avi video/
```

vs.

```
$ cp foo.avi video/
$ rm foo.avi
```

A. oui B. non

Si les fichiers sont sur le même système de fichier:

- `mv` ne fait que créer un nouveau nom / déplacer le nom, sans déplacer les données
- `cp + rm` copie les données puis les supprime (beaucoup moins efficace)

AGIR SUR UN GROUPE DE FICHIER WILDCARDS

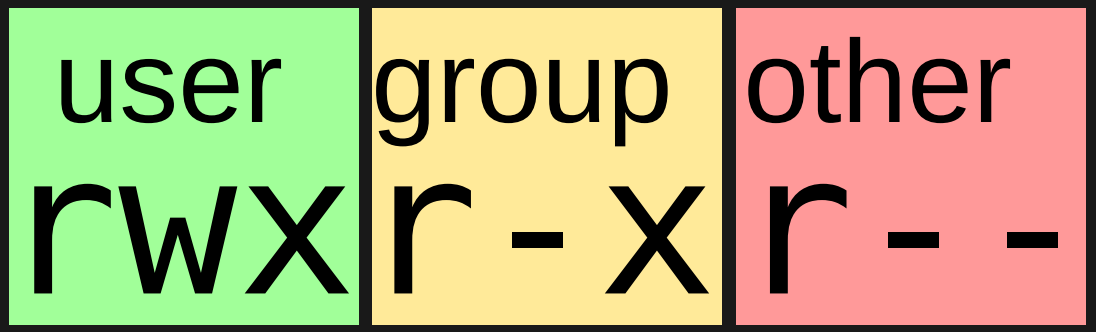
- On peut utiliser les métacaractères (*wildcards*) suivants dans les noms des fichiers:
  - `*` remplace zéro, un ou plusieurs caractères
  - `?` un seul caractère
- Si le nom du fichier contient un astérisque ou un point d'intérogation on peut utiliser un backslash pour l'échapper.

LE CAUCHEMAR

Attention !

```
$ rm -rf ~/*.txt
$ rm -rf ~/* .txt
```

PERMISSIONS



read

write

execute

PERMISSIONS - EFFETS

Fichier:

Read	Lire le contenu du fichier
Write	Modifer le contenu d'un fichier Warning en cas de suppression
eXecute	Exécuter le fichier (exécutable binaire ou script)

Répertoire:

Read	Lister les **noms** des fichiers inclus (mais pas les métadonnées)
Write	Créer, renommer ou détruire les fichiers inclus
eXecute	"Ouvrir" le répertoire, voir les métadonnées, accèder au contenu des fichiers (mais pas aux noms), exécuter les fichiers exécutables.

PERMISSIONS - OCTAL (3)

user	group	other
rwx	r-x	r--

read = 4
write = 2
execute = 1

rwx	r-x	r--		
421	401	400	=>	754
7	5	4		

CHANGER LES PERMISSIONS (chmod)

```
chmod [options]... permission fichiers...
```

L'option -R agit récursivement sur le contenu des répertoires

```
$ chmod 666 foo.txt
$ chmod 640 videos/*
$ chmod 750 videos/
$ chmod -R 750 videos/  # change les permissions de videos/ mais aussi de TOUT son contenu
```

CHANGER LES PERMISSIONS (2)

- La command `chmod` accepte également une représentation symbolique:
- Elle se forme de la manière suivante:
  1. u (user), g (group), o (other), ou a (all)
  2. + (add), - (remove), ou = (set)
  3. Les permission au "format" `rwx`

```
$ chmod a+rw foo.txt  # ajoute les droit read et write à tous
$ chmod u=w foo.txt   # l'utilisateur n'a QUE le droit w
$ chmod g-wx foo.txt  # le groupe perd les droit write et execution
```

CHANGER LE GROUPE (`chgrp`)

```
chrgp [options]... groupe fichiers...
```

L'option `-R` agit récursivement sur le contenu des répertoires

```
$ chgrp video *.avi
$ chgrp -R admin notes/
```

FICHIERS CACHÉS

Il suffit d'ajouter un point au début du nom d'un fichier pour le cacher.

```
$ mv foo.txt .foo.txt
$ cp ~/.bashrc /tmp/bashrc
```

AUTRES COMMANDES UTILES

1. Afficher l'espace restant sur toutes les partitions:

```
$ df -h
```

2. Calculer la taille de chaque sous-répertoire et fichier:

```
$ du -sh *
```

3. Chercher récursivement par le nom:

```
$ find . -name "*.txt"
```

4. Créer un lien symbolique:

```
$ ln -s /machin/chose/truc/ ~/truc
```

EVERYTHING IS A FILE

- Plusieurs concepts sont représentés par des **fichiers synthétiques**.
- Par exemple:
  - Les périphériques sont visibles dans `/dev/`
  - Les processus (et certaines info système) sont visibles dans `/proc/`
  - Informations sur les périphériques dans `/sys/`

EXEMPLES D'UTILISATION DE /dev/

Faire une image ISO à partir d'un CD

```
$ dd if=/dev/cdrom of=~/image.iso
```

Remplir une partition de données pseudo-aléatoires

```
$ dd if=/dev/urandom of=/dev/sdb1 bs=1M
```

FLUX ET REDIRECTIONS

FLUX DE SORTIE STANDARD

- La plupart des commandes (des processus) ont un flux de sortie.
- Par défaut ce flux est dirigé vers le shell.
- On peut le rediriger vers un fichier avec >:
  - `commande arguments > fichier`

CONCATÉNER (`cat`)

On peut concaténer plusieurs fichiers avec `cat`

```
cat [options]... [fichiers]...
```

Le résultat est envoyé sur le flux de sortie

Options:

-n numérote les lignes

-b numérote les lignes non-vides

```
$ cat foo.txt
$ cat foo.txt bar.txt
$ cat foo.txt bar.txt > all.txt  # ecrit le resultat dans all.txt au lieu su shell
$ cat episode1.avi episode2.avi episode3.avi > full.avi  # est-ce que ca marche ?
```

SAUVER LE RÉSULTAT D'UNE COMMANDE

On peut sauver le résultat de toute commande utilisant le flux de sortie:

```
$ ls -lh > content.txt
$ df -h > disk_usage.txt
$ w > users.txt
```

FLUX D'ENTRÉE STANDARD

- La plupart des commandes (des processus) ont un flux d'entrée.
- Par défaut ce flux est est issu du clavier
- On peut le rediriger vers un fichier avec <:
  - `commande arguments < fichier`

COMPTER (wc)

On peut compter les caractères/mots/lignes avec wc

```
wc [options]... [fichiers]...
```

- Le résultat est envoyé sur le flux de sortie
- Par défaut, l'entrée standard est utilisée

Options:

-c compte les caractères

-w compte les mots

-l compte les lignes

```
$ wc -w foo.txt
$ wc -w < foo.txt
$ wc -w < foo.txt > words.dat
$ wc -w
```

ENTRÉE STANDARD

On peut interrompre le flux avec *ctrl+D*.

```
$ cat > foo.txt
```

AJOUT (>>)

- La redirection > écrase la destination si elle existe
- On peut utiliser >> pour ajouter à un fichier existant

```
$ wc -l < foo.txt >> stats.dat
$ wc -l < bar.txt >> stats.dat
$ wc -l < baz.txt >> stats.dat
```

PIPES (TUBES)

On veut concaténer des fichiers et compter les lignes du résultat:

```
$ cat foo.txt bar.txt baz.txt > all.txt
$ wc -l < all.txt > stats.dat
```

On peut utiliser les *pipes* (|) pour rediriger la sortie d'une commande vers l'entrée de la suivante:

```
$ cat foo.txt bar.txt baz.txt | wc -l > stats.dat
```

TRIER LES LIGNES (`sort`)

```
sort [options]...
```

Options:

-r	trie "à l'envers"
-n	tri numérique
-h	tri des représentations "humaines"

```
$ du -sh * | sort -h
$ du -sh * | sort -rh
$ sort < words.txt > sorted.txt
```

GARDER LES PREMIÈRES LIGNES (head)

```
head [options]... [fichiers]...
```

Options:

Par défaut garde les 10 premières lignes

-n *k*	garde les *k* permières lignes
-c *k*	garde les *k* premiers bytes

```
$ head -n 5 < foo.txt
$ du -sh * | sort -rh | head -n 5 > biggest.txt
```

FLUX D'ERREUR STANDARD

- La plupart des commandes (des processus) ont un flux d'erreur.
- **Par défaut** ce flux est dirigé vers **la console**.
- On peut le rediriger vers un fichier avec 2>:
  - `commande arguments 2> fichier`
- On peut rediriger le flux d'erreur vers le flux de sortie avec 2>&1:
  - `commande arguments > fichier 2>&1`

FLUX STANDARD DANS LES LANGAGES DE PROGRAMMATION

	C	*C++*	*Java*	*Python*	*Ruby*
Entrée	stdin	std::cin	System.in	sys.stdin	$stdin
Sortie	stdput	std::cout	System.out	sys.stdout	$stdout
Erreur	stderr	std::cerr	System.err	sys.stderr	$stderr

Exemple

```
System.out.println( "Hello world!" );
```

Redirection

```
$ java HelloWorld > hello.txt
```

PROCESSUS

PROCESSUS (1)

- Instance d'un programme en cours d'execution
- Possède:
  - Son code en langage machine
  - Des Segments de mémoire
  - Des descripteurs de fichiers
  - Un propriétaire et un ensemble de permissions
  - Un état à un moment donné
- Un coeur de processeur ne peut exécuter qu'un processus à la fois
- Le système d'exploitation peut alterner l'execution de plusieurs processus (*multi-tasking*)

PROCESSUS (2)

Sous unix, tous les processus ont:

- un identifiant numérique (*PID*)
- un processus parent (sauf le processus 0)
- une priorité

AFFICHER LES PROCESSUS (ps)

```
ps [options]...
```

Exemples d'options:

-eF	Voir tous les processus
-ejH	Voir l'arbre des processus
-u falcone	Voir tous les processus de l'utilisateur `falcone`

INTERLUDE: `grep`

La commande `grep` est extrememement utile car elle permet de **filtrer les lignes qui contiennent un certain motif**

```
grep [options]... <motif> [fichiers]...
```

Options utiles:

-i	ignore les majuscules
-v	exclus les lignes qui contiennent le motif
-r	cherche recursivement (sous-répertoires)

```
$ grep falcone /etc/passwd
$ ps -eF | grep java
```

UTILISATION DU CPU (`top`)

La commande `top` permet de voir quels processus occupent le(s) CPU(s).

SIGNAUX

- Les signaux permettent une communication limitée entre les processus.
- La commande `kill -l` permet d'afficher les différent signaux
- Quelques signaux utiles:

Nom	Action	Numero
SIGINT	Interrompt le processus	2
SIGKILL	Tue le processus immédiatement	9
SIGTERM	Demande la terminaison (propre)	15
SIGTSTP	Suspension	20
SIGCONT	Active un processus suspendu	18
SIGUSR1	Signal utilisateur 1	10
SIGUSR2	Signal utilisateur 2	12

ENVOYER UN SIGNAL (`kill`)

```
kill -signal [pids]...
```

`kill` permet d'envoyer un signal à un processus

```
$ kill -SIGKILL 2111 2120
$ kill -15 2111
```

GESTION DES SIGNAUX PAR LES PROCESSUS

- Tous les processus peuvent gérer (intercepter) un signal.
- Par exemple:
  - Sauver l'état avant de quitter lorsque `SIGTERM` est reçu
  - Redémarer le processus lorsque `SIGINT` est reçu
- Le signal `SIGKILL` ne peut être intercepté

SIGNAUX DEPUIS LE SHELL

Les raccourcis claviers suivant permettent de lancer un signal à un processus du shell:

ctrl+C	Envoie `SIGINT`
ctrl+Z	Envoie `SIGTSTP`

Attention ctl+D n'est pas un signal mais envoie "End-Of-File" (EOF) sur l'entrée standard

JOBS

- Par défaut, tout processus lancé depuis le shell tourne à l'avant-plan (*foreground*)
- Un processus en avant-plan bloque le shell tant qu'il n'est pas fini.
- On peu aussi lancer un processus en arrière-plan (*background*):
  - Ajouter un & après la commande:

```
$ gedit &
```

  - Interrompre un processus:
    - Utilisez `ctrl-Z` pour interrompre le processus
    - Utilisez la commande `bg` pour relancer un processus interrompu en arrière-plan
    - On peut aussi utiliser `fg` pour remettre le dernier processus interrompu en avant-plan

RESSOURCES

http://www.ibm.com/developerworks/aix/library/au-speakingunix8/