Le Système Unix

Jean-Luc Falcone

Septembre 2018

Interface texte

- Pourquoi utiliser une interface texte en 2018 ?

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

```
henni@henni:~
henni     6941  6911  0 12:47 ?        00:00:00 kde-window-decorator -session 10d1656e6e00012070574670
henni     6944     1  0 12:47 ?        00:00:11 yakuake -session 10d1656e6e0001207168808000008516002 9
henni     6947  6911  0 12:47 ?        00:00:00 konqueror [kdeinit] --preload
henni     6952     1  0 12:47 ?        00:00:00 klipper [kdeinit]
henni     6954     1  0 12:47 ?        00:00:00 kalarmd --autostart
henni     6959     1  0 12:47 ?        00:00:00 knotify [kdeinit]
henni     6967  6911  0 12:47 ?        00:00:00 konqueror [kdeinit] --preload
henni     6969  6911  0 12:47 ?        00:00:00 ksystraycmd /usr/bin/thunderbird-bin
henni     6970  6969  0 12:47 ?        00:00:00 /bin/bash /usr/libexec/mozilla-launcher
henni     6980  6970  0 12:47 ?        00:00:29 /opt/thunderbird/thunderbird-bin
henni     6984     1  0 12:47 ?        00:00:00 /usr/libexec/gconfd-2 19
henni     7083     1  0 12:52 ?        00:00:00 /usr/bin/gpg-agent --sh --daemon --write-env-file /hom
henni     7671  6911  0 16:15 ?        00:00:00 kio_file [kdeinit] file /home/henni/.kde3.5/socket-hen
henni     7677  6918  0 16:15 ?        00:00:01 konqueror /home/henni
henni     7711     1  1 16:16 ?        00:01:06 amarokapp
henni     7723  7711  0 16:16 ?        00:00:00 ruby /usr/share/apps/amarok/scripts/score_default/scor
henni     7724  7711  0 16:16 ?        00:00:00 ruby /usr/share/apps/amarok/scripts/lyrics_lyrc/lyrics
henni     7725  7711  0 16:16 ?        00:00:00 /usr/bin/python /home/henni/.kde3.5/share/apps/amarok/
henni    18419     1  0 14:04 ?        00:00:00 kio_uiserver [kdeinit]
henni    19409  6911  0 17:20 ?        00:00:00 kio_http [kdeinit] http /home/henni/.kde3.5/socket-hen
henni    19716  6944  0 17:28 pts/1    00:00:00 /bin/bash
henni    20046  6918  0 17:37 ?        00:00:00 /usr/kde/current/bin/ksnapshot
henni    20050  6911  0 17:37 ?        00:00:00 kio_file [kdeinit] file /home/henni/.kde3.5/socket-hen
henni    20077 19716  2 17:37 pts/1    00:00:00 xterm
henni    20079 20077  4 17:37 pts/2    00:00:00 bash
henni    20090 20079  0 17:37 pts/2    00:00:00 ps -ef
henni@henni ~ $ aoue
```

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:

- Restreindre l'accès à un répertoire à un seul groupe
- Permettre l'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

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 !

Système de fichier (1)

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

Systèmes de fichier (2)

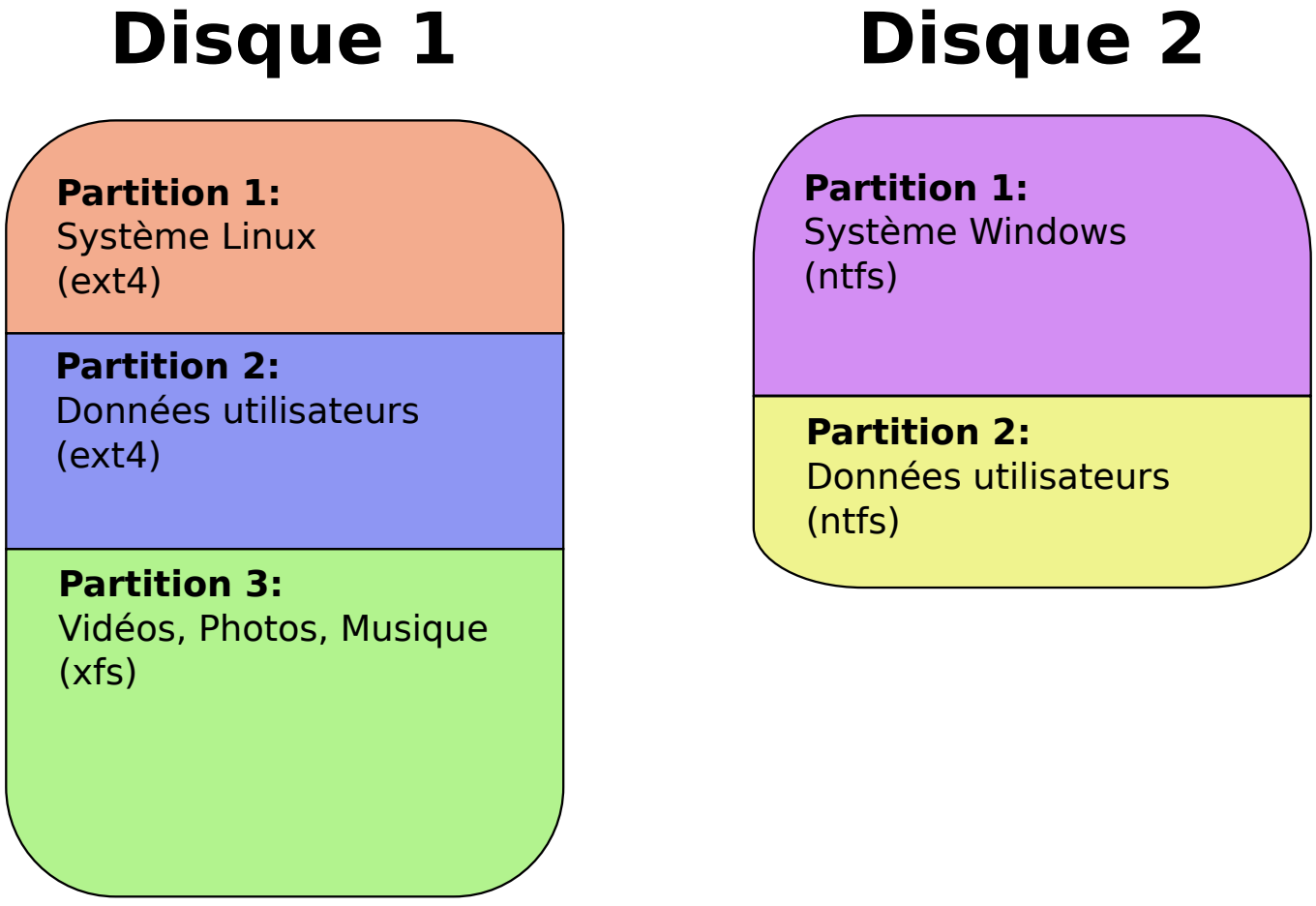
- Il existe plusieurs types de systèmes de fichiers. Par exemple:

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

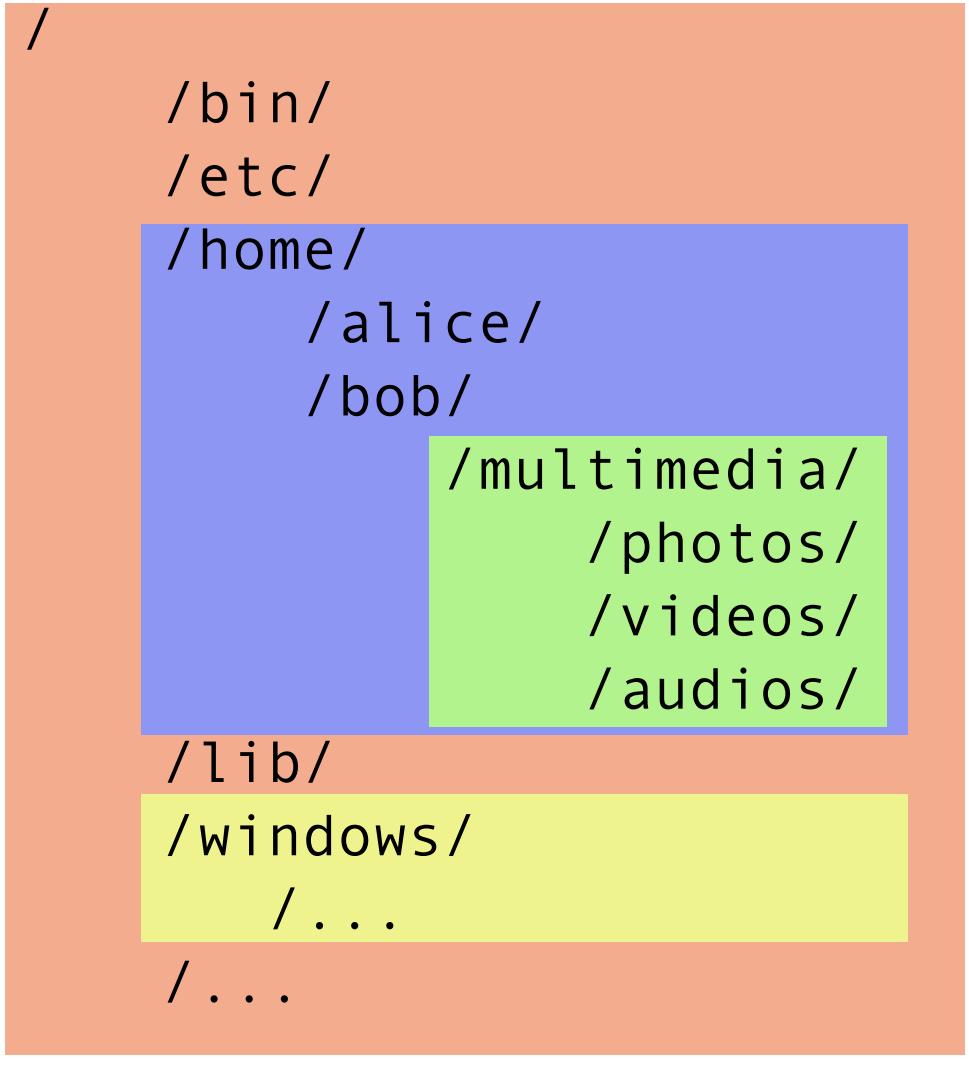
Arborescence (1)

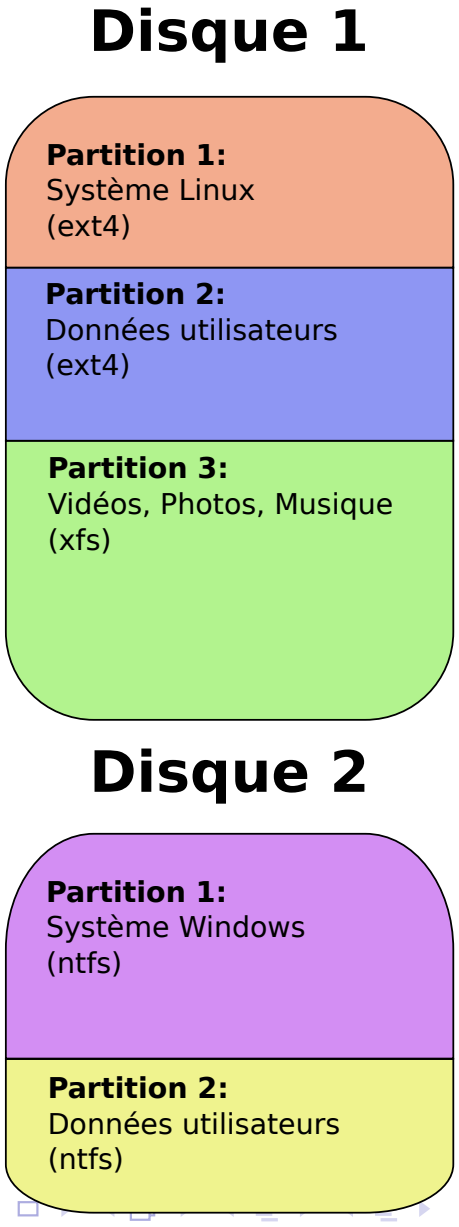
- Tous les systèmes de fichiers sont "montés" sur une seule et même arborescence.
- Seul `root` peut monter/démonter des systèmes de fichiers (par défaut)
- La plupart des installations GNU/Linux actuelles, montent automatiquement les disques externes (CD, DVD, usb)

Arborescence (2)



Arborescence (3)





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

Commande `ls`

- Commande: `ls [options] [répertoire]`
- 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
- Si la sortie est longue:
  - `ls -l | more`
  - `ls -l | less`

Créer un repertoire (`mkdir`)

- Commande: `mkdir [options] <repertoire à créer>`
- L'option `-p` permet de créer automatiquement les repertoires parents.

Exemples

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

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

Déplacer des fichiers (mv)

- Commande: `mv [options] <fichiers> <destination>`
- Fonctionne aussi avec des répertoires

Exemples

```
$ mv foo.avi bar.mpg baz.mkv video/
$ mv SysInfo/ InfoBio ~/cours
```

Renommer des fichiers (mv)

- Commande: `mv [options] <nom avant> <nom après>`

Exemples

```
$ mv foo.AVI foo.avi
$ mkdir bar
$ mv bar/ foo/
```

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)

- Commande: `cp [options] <fichiers> <destination>`

Exemples

```
$ cp foo.txt foo.txt.backup
$ cp bar.txt baz.doc projet/
```

Options de la commande cp

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

- `-f` écrase 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*)

Autre options essentielles:

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

Effacer un repertoire (`rmdir`)

- Commande: `rmdir <repertoire>`
- Fonctionne uniquement avec un répertoire vide.

Exemples

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

Effacer des fichiers (`rm`)

- Commande: `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*)

Exemples

```
$ rm foo.txt foo.txt.backup
$ rm -rf foo/
```

mv vs. cp/rm

Y'a-t-il une différence ?

```
$ mv foo.avi video/

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

Agir sur un groupe de fichier wildcards

Blah

- 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.

Exemples

```
$ mv *.png *.jpg images/
$ cp data00?.txt backup/
$ rm -f report*2012.*
$ rm got_cash\?.doc
```

Le Cauchemar

Attention !

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

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

Permissions (1)

user	group	other
rwx	r-x	r--

read
write
execute

Permissions (2)

- Fichier:
  - Read: Lire le contenu du fichier
  - Write: Modifer le contenu d'un fichier
  - 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 (3)

user	group	other
rwx	r-x	r--

read = 4
write = 2
execute = 1

rwx	r-x	r--
421	401	400
7	5	4

=> 754

Changer les permissions d'un fichier (chmod)

- Commande: `chmod [options] <permission> <fichiers>`
- L'option `-R` agit récursivement sur le contenu des répertoires

Exemples

```
$ chmod 666 foo.txt
$ chmod 640 videos/*
$ chmod 750 videos/
```

Changer les permissions d'un fichier (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`

Exemples

```
$ chmod a=rwx foo.txt
$ chmod -R g+r videos/
$ chmod -R g-w videos/
$ chmod -R o-rwx videos/
```

Changer le groupe d'un fichier (chgrp)

- Commande: `chrgp [options] <groupe> <fichiers>`
- L'option `-R` agit récursivement sur le contenu des répertoires

Exemples

```
$ 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.

Exemples

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

Autres commandes utiles

Afficher l'espace restant sur toutes les partitions:

```
$ df -h
```

Calculer la taille de chaque sous-répertoire et fichier:

```
$ du -sh *
```

Chercher récursivement par le nom:

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

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 de Sortie Standard

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

Concaténer (cat)

- On peut concaténer plusieurs fichiers avec `cat`
- Commande: `cat [options] [fichiers]`
- Le résultat est envoyé sur le flux de sortie
- Options:
  - `-n` numérote les lignes
  - `-b` numérote les lignes non-vides

Exemples

```
$ cat foo.txt
$ cat foo.txt bar.txt
$ cat foo.txt bar.txt > all.txt
$ cat episode1.avi episode2.avi episode3.avi > full.avi
```

Sauver le résultat d'une commande

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

Exemples

```
$ 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`
- Commande: `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

Exemples

```
$ 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*.

Ecrire un petit fichier:

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

Ajout (»)

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

Exemples

```
$ 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`)

- Commande: `sort [options]`
- Options:
  - `-r` trie "à l'envers"
  - `-n` tri numérique
  - `-h` tri des représentations "humaines"

Exemples

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

Garder les premières lignes (`head`)

- Commande: `head [options] [fichiers]`
- Par défaut garde les 10 premières lignes
- Options:

  -n *k* garde les *k* permières lignes
  -c *k* garde les *k* premiersbytes

Exemples

```
$ 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

Exemples

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

Redirection

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

Processus (1)

- Instance d'un programme en cours d'execution
- Possède:
  - Son code en langage machine
  - Un Segment 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)

- Syntaxe: `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.
- Elle permet de filtrer les lignes qui contiennent un certain motif
- Syntaxe: `grep [options] <motif> [fichiers]`
- Options utiles:
  - -i ignore les majuscules
  - -v exclus les lignes qui contiennent le motif
  - -r cherche recursivement (sous-répertoires)

  :BEAMER$_{env}$: block

```
$ 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:

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`)

- La commande `kill` permet d'envoyer un signal
- Syntaxe: `kill <-signal> <pids>`

Exemples

```
$ kill -9 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`

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 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/