

# Linux de Base

Merci à Arnaud dupont

# Systeme d'exploitation

---



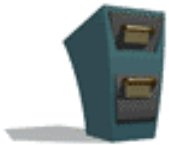
Un système d'exploitation exploite quoi ?

# Les systèmes d'exploitation

---

- *introduction* -

- C'est l'interface entre l'utilisateur et le matériel
- Ses fonctions principales sont :
  - Contrôle des ressources (allocation et gestion du CPU et de la mémoire)
  - Contrôle des processus
  - Contrôle des périphériques
  - ...



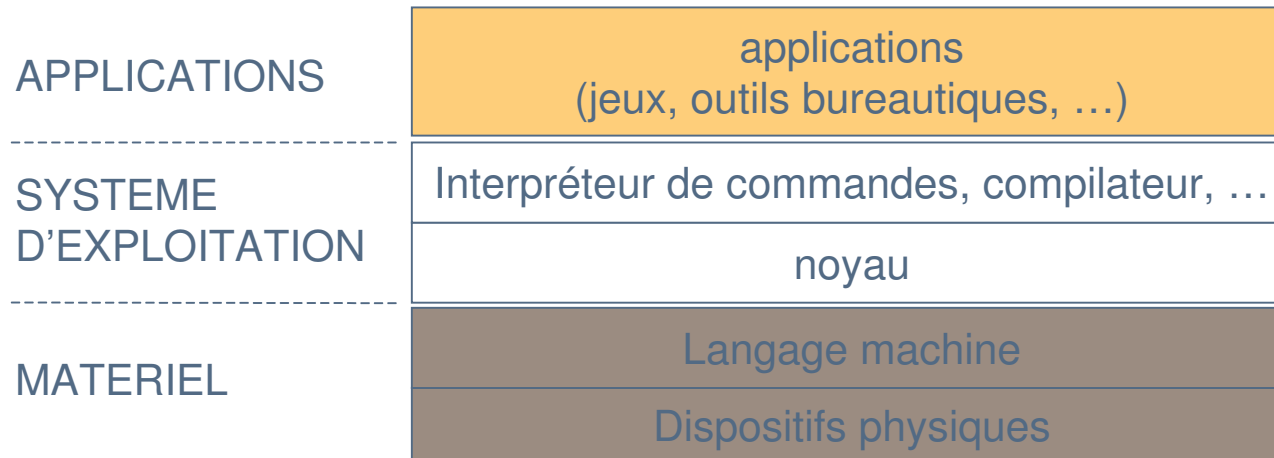
Il contient des outils de gestion utilisables par les applications, tels que la manipulation de fichiers, gestion d'impressions, date...

# Les systèmes d'exploitation

---

- *introduction* -

- Exemples:
  - Unix, DOS, Windows, Mac OS, Linux, OS/2, BSD, ...
- Architecture-type:



# Linux: on y va !

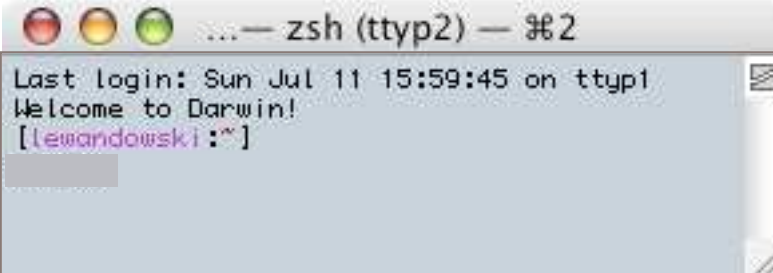
---

- Propriétés
  - multi-tâches
  - multi-utilisateurs
  - multi-postes
  - Libre (et gratuit) !!
- Ouverture/Fermeture d'une session
  - Travailler sous Linux implique une connexion au système
  - Login:
    - Identification de l'utilisateur: *login + mot-de-passe*
    - droits accordés par le *super-utilisateur (root)*
  - Logout:
    - **NE PAS ETEINDRE** une machine "sauvagement"
    - commande "logout" dans la console

# Initiation au shell

---

- Une fois connecté, le système nous connaît, ouvre une session à notre nom et attend nos instructions via un programme spécial:
- Le Shell = interpréteur de commandes
  - interface utilisateur “de base” (interlocuteur avec le syst.)
  - interprétation ligne à ligne
  - plusieurs shells: sh, csh, tcsh, bash, ksh, zsh, ...
  - configurable: fichiers d’environnement (commençant par un “.”)
    - “.login”
    - “.logout”
    - “.bashrc”
  - langage de programmation
- shell par défaut : bash



```
... — zsh (ttyp2) — #2
Last login: Sun Jul 11 15:59:45 on ttyp1
Welcome to Darwin!
[lewandowski :~]
```

# Initiation au shell

---

- *commandes* -

- Format des commandes:

▶▶▶ `cde [-option(s)] [argument(s)]`

# Initiation au shell

- *commandes* -

- Exemples:

- **date**

- **whoami**

- affiche le nom de l'utilisateur connecté

- **echo**

- affiche un message (echo "bonjour !")

- **ls**

- liste le contenu  
d'un répertoire

- **man <cde>**

- manuel en ligne



```
Terminal — zsh (tty1) — %1
[lewindow:~] ls -l
total 0
drwx-----  4 lewindow  staff    136 11 Jul 18:53 Desktop
drwx----- 17 lewindow  staff    578  9 Jul 10:22 Documents
drwx----- 59 lewindow  staff   2006  9 Jul 16:25 Library
drwxr-xr-x 12 lewindow  staff    408  7 Jul 14:50 Movies
drwxr-xr-x 17 lewindow  unknown  578  2 Jul 15:36 Music
drwx----- 11 lewindow  staff    374  7 Jul 15:38 Pictures
drwxr-xr-x  3 lewindow  staff    102 18 Jun 13:16 Public
drwxr-xr-x 19 lewindow  staff    646  9 Jul 10:39 boulot
drwxr-xr-x 47 lewindow  staff   1598 12 Jul 09:32 downloads
drwxr-xr-x  9 lewindow  staff    306  9 Jul 09:42 utils
[lewindow:~]
```



# Initiation au shell

---

- *méta caractères* -

- Caractères spéciaux:

! ^ \* ? [ ] \ ; & < > | >>

- ▶▶▶ • L'astérisque ou étoile: \*

  - interprété comme toute suite de caractères alphanumériques
  - utiliser **avec précaution** (commande rm par ex...)

- Le point d'interrogation: ?
  - remplace 1 seul caractère alphanumérique

# Initiation au shell

---

## - *méta caractères* -

- Le point-virgule: `;`
  - Séparateur de commandes
- Les crochets: `[]`
  - Remplace un caractère choisi parmi ceux énumérés entre les crochets
- L'anti-slash: `\`
  - Inhibe la signification du méta-caractère qui suit
- Interprétation des chaînes de caractères
  - Texte entre `' '` (simples quotes): le texte n'est pas interprété mais considéré comme un mot
  - Texte entre `" "` (doubles quotes): seuls sont interprétés les métacaractères `$`, `\` et ```
  - Texte entre `` `` (anti quotes): considéré comme une commande à interpréter, et c'est le résultat qui sera utilisé.

# Initiation au shell

---

## - *méta caractères* -

- Exemples:
  - **echo \***  
Tous les fichiers sauf ceux dont le nom commence par un point
  - **echo \*c**  
Tous les fichiers dont le nom se termine par un 'c'
  - **echo .\***  
Tous les fichiers dont le nom commence par un point
  - **echo [0-9]\***  
Tous les fichiers dont le nom commence par un chiffre



```
...inal — zsh (ttyp2) — ⌘2
[lewandow:~] echo 'date'
date
[lewandow:~]
[lewandow:~]
[lewandow:~] echo `date`
Mon Jul 19 17:59:10 CEST 2004
[lewandow:~]
[lewandow:~]
[lewandow:~] echo "date = `date`"
date = Mon Jul 19 17:59:19 CEST 2004
[lewandow:~]
[lewandow:~]
[lewandow:~]
```

# Le système de fichiers

---

- Stocke les données:
  - de façon hiérarchique
  - structure arborescente
  - TOUT est fichier
- 3 types de fichiers:
  - fichiers ordinaires
  - répertoires
  - fichiers spéciaux (périph., ...)

# Le système de fichiers

---

- *fichiers* -

- Accès aux fichiers réglementé (sauf: tous les droits pour **root**)
- 3 types d'utilisateurs:
  - propriétaire (**user**)
  - personnes du mm groupe (**group**)
  - les autres (**others**)
- 3 types de permissions

– lecture ( <b>r</b> )	afficher le contenu	afficher le contenu
– écriture ( <b>w</b> )	modifier	créer/supp fichiers
– exécution ( <b>x</b> )	exécuter	traverser
	<b>fichier</b>	<b>répertoire</b>

# Le système de fichiers

---

- fichiers -

- Affichage des caractéristiques: **ls -l**

`-rw-r--r--` 1 dupont staff 58K 16 Jul 09:19 tp1.tex

nb liens      propriétaire      taille      date      nom

groupe

type      group

`-rw-r--r--`

user      others

# Le système de fichiers

---

## - fichiers -

- Changer les permissions: **chmod**

**chmod <classe op perm, ...> | nnn <fic>**

– classe:

u : user

g : group

o : others

a : all

– op:

= : affectation

- : suppr.

+ : ajout

– perm:

r : lecture

w : écriture

x : exécution

– chaque perm = 1 valeur:

r	4
w	2
x	1
rien	0

– déf. des permissions (par addition)  
pour chaque classe

exemples:

```
chmod u=rwx,g=rx,o=r tp1.tex
```

```
chmod a+x script.sh
```

```
chmod 755 script.sh
```

# Le système de fichiers

---

- *fichiers* -

- Manipulation des fichiers
  - copier : `cp fic1 fic2`
  - déplacer/renommer : `mv fic1 fic2`
  - effacer : `rm fic`
  - afficher le contenu : `cat fic`
  - trier le contenu : `sort fic`

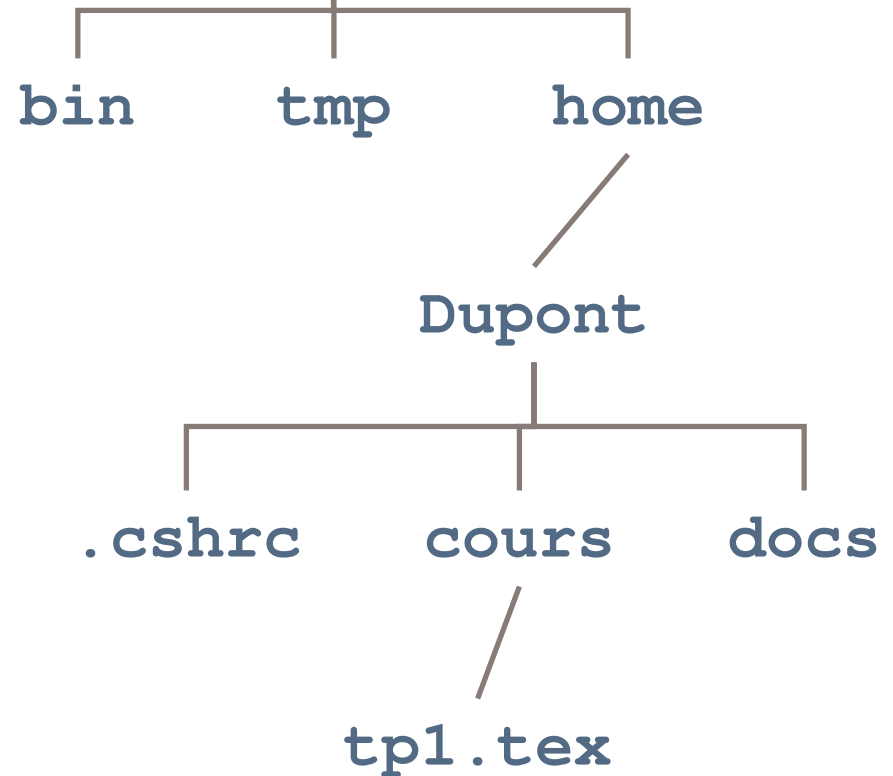
Voir les pages du “man” !!



# Le système de fichiers

- l'arborescence -

répertoire racine



- le répertoire de login: `~`
- le répertoire courant: `.`
- le répertoire supérieur: `..`
- connaître le rép. courant: `pwd`
- lister le contenu: `ls`  
(voir "man ls")

- chemin d'accès au fichier `tp1.tex`:
  - `/home/ldupont/cours/tp1.tex`
  - ou bien: `~/cours/tp1.ex`

# Le système de fichiers

- **pwd** retourne:  
`/home/dupont/cours`

- se déplacer: **cd**

```
[/home/dupont/cours]$ cd ..
```

```
[/home/dupont]$
```

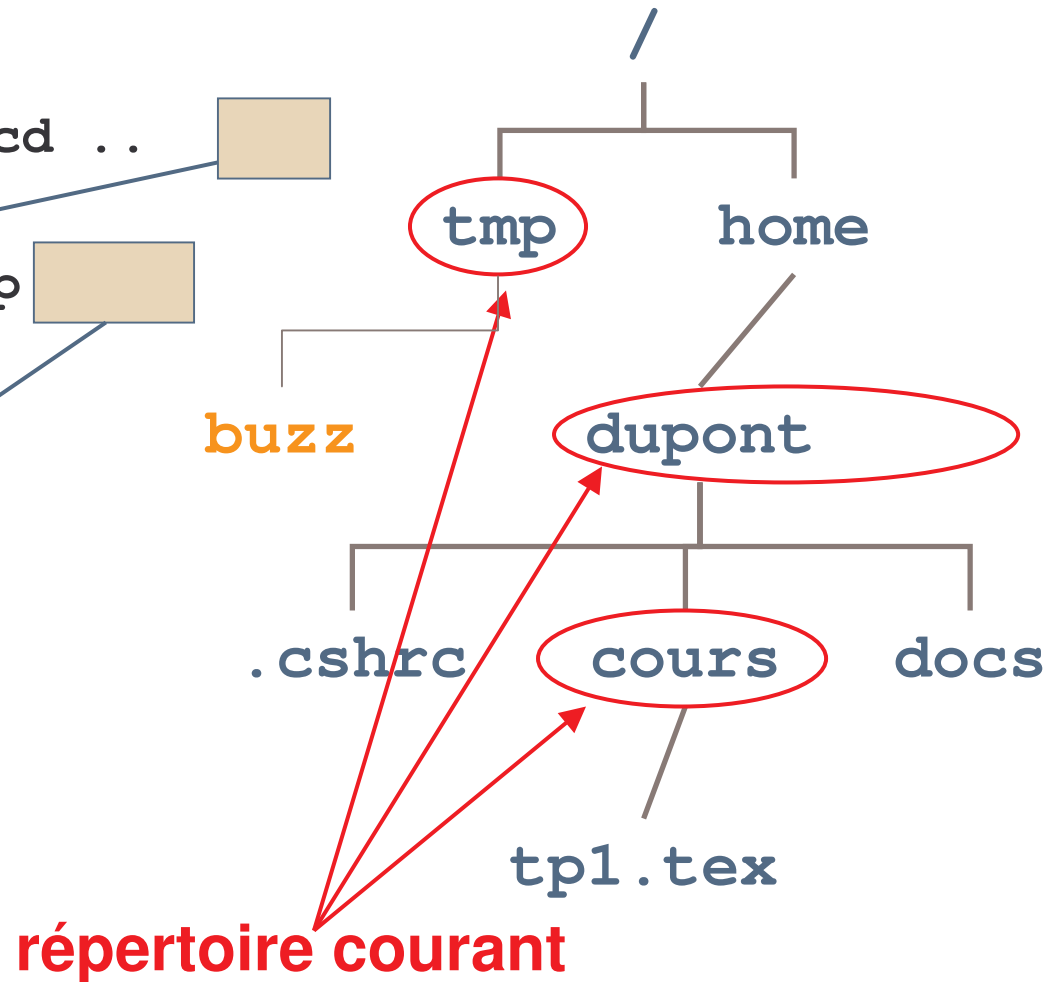
```
[/home/dupont]$ cd /tmp
```

```
[/tmp]$
```

- chemin relatif
- chemin absolu

- créer un répertoire: **mkdir**  
`[/tmp]$ mkdir buzz`
- supprimer un répertoire: **rmdir**  
`[/tmp]$ rmdir buzz`

- l'arborescence -



# Le système de fichiers

---

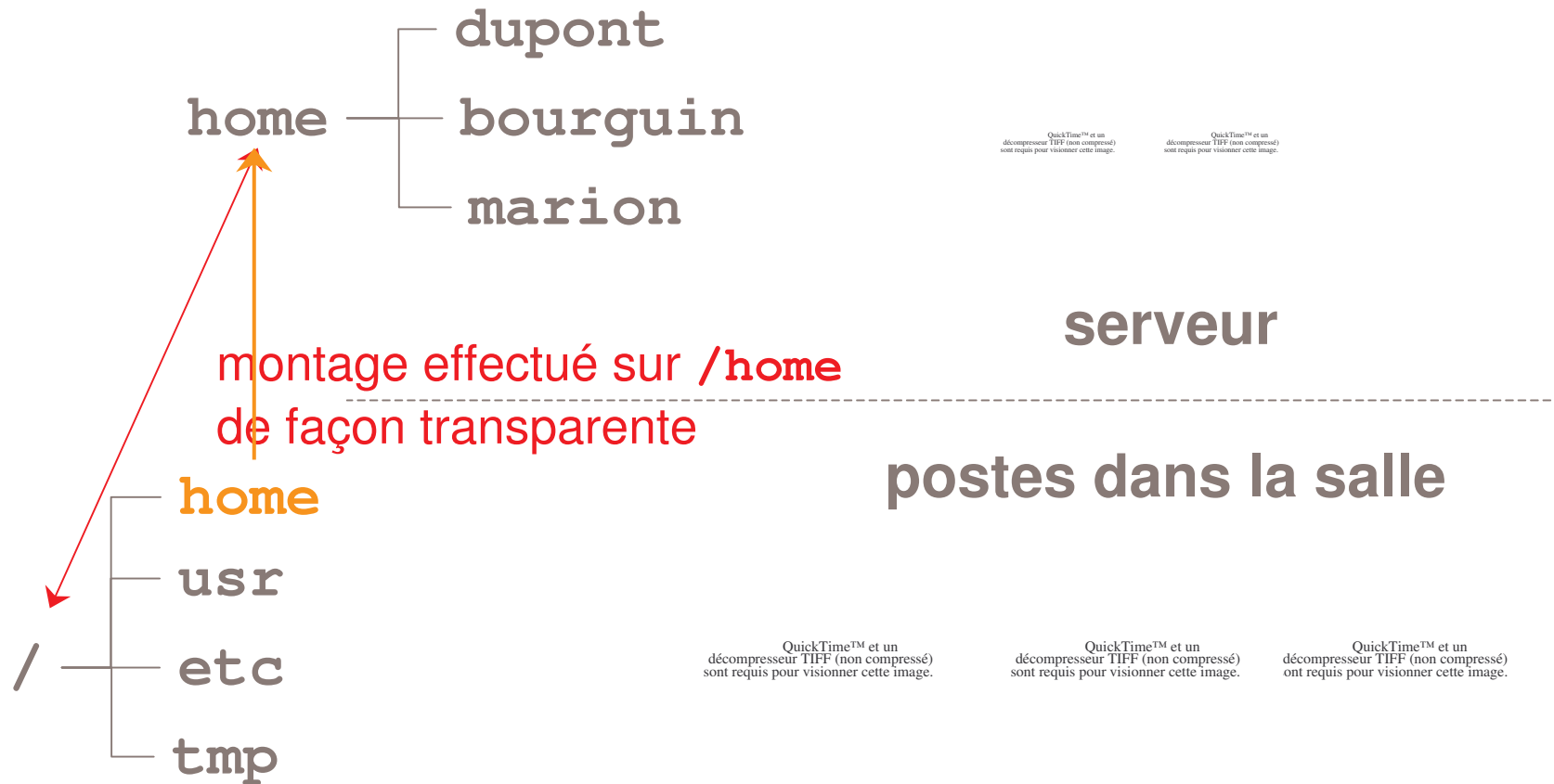
- *partitions* -

- le disque est “découpé” en partitions
  - commandes & applications
  - comptes utilisateurs
  - swap
  - fichiers temp
  - périphériques (disques, ...)
  - ...
- accès transparent

# Le système de fichiers

- partitions -

- exemple: les comptes utilisateurs



# Le système de fichiers

---

- *partitions* -

- tous les disques amovibles (disquette, cdrom, clé usb) dans : **/mnt**
- ex pour utiliser une disquette:
  - Montage:  
**mount /mnt/floppy**
  - lire/écrire dans **/mnt/floppy**
  - Démontage:  
**umount /mnt/floppy**
- idem pour clés usb

# Le système de fichiers

---

## - liens -

- Liens physiques

`ln <nom_fic> <nouveau_nom_fic>`

- permet de donner plusieurs noms à un fichier
- pas pour les répertoires
- ne traverse pas les partitions
- un fic est détruit quand TOUS ses liens physiques sont supprimés (≠ raccourcis)

- Liens symboliques

`ln -s <nom_fic> <nouveau_nom_fic>`

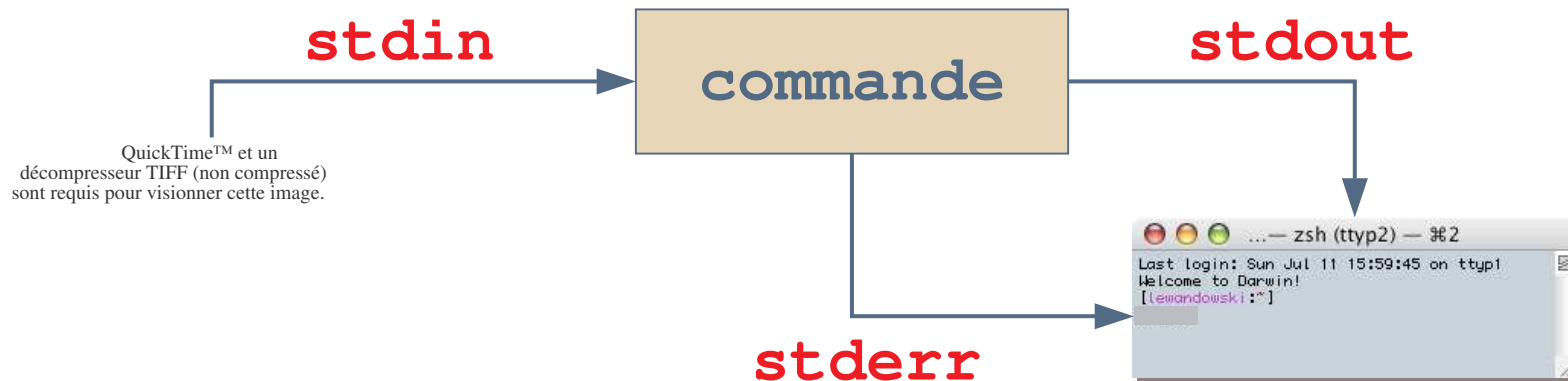
- crée un **raccourci**
- traverse les partitions
- fonctionne aussi pour les répertoires

- Lister les liens d'un fichier: `ls -l <nom_fic>`

# Les redirections

---

- Une commande ouvre 3 descripteurs de fichiers; par défaut:



- Redirections= remplacer les canaux par défaut, rediriger vers une autre commande ou un fichier

# Les redirections

---

<	redirige l'entrée standard
>	redirige la sortie standard
>>	concatène la sortie standard
2>	redirige la sortie d'erreur
&>	redirige la sortie standard <b>et</b> la sortie d'erreur

exemples:

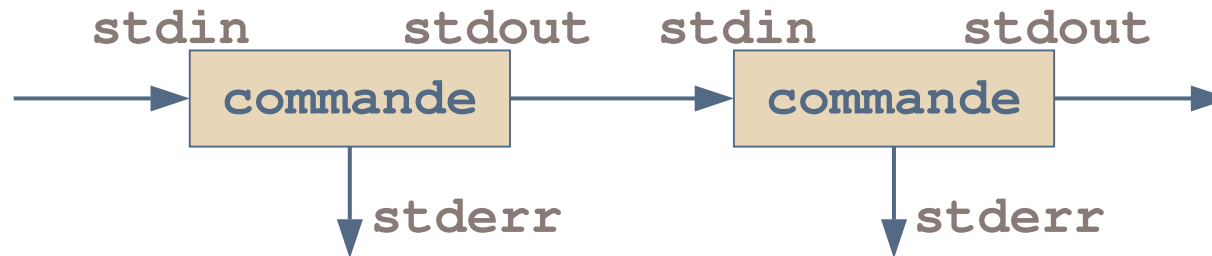
<code>ls . &gt; liste</code>	crée/écrase le fichier liste et y dirige la sortie de 'ls'
<code>date &gt;&gt; liste</code>	ajoute à la fin du fichier liste la sortie de 'date'
<code>wc -l &lt; liste</code>	envoie comme entrée à la commande 'wc' le fichier liste



# Les tubes (pipes)

---

- Tube: |
- pour “connecter 2 commandes”



ex: combien de fichiers dans le rep. courant ?

sans pipe:

```
ls > temp ; wc -l < temp ; rm temp
```

avec un pipe:

```
ls | wc -l
```

# Les processus

---

- Processus = objet dynamique qui représente un programme en cours d'exécution et son contexte
- Caractéristiques:
  - identification (pid)
  - identification du proc. parent (ppid)
  - propriétaire
  - priorité
  - ...
- Pour voir les processus en cours: **ps**

# Les processus

- Infos retournées par `ps`:

	PID	TT	STAT	TIME	COMMAND	
	3899	p1	S	0:00.08	-zsh	
	4743	p1	S+	0:00.14	emacs	
	4180	std	S	0:00.04	-zsh	

**numéro de processus** → PID

**terminal associé** → TT

**temps CPU utilisé** → TIME

**commande exécutée** → COMMAND

**état du processus:**

R	actif
T	bloqué
P	en attente de page
D	en attente de disque
S	endormi
IW	swappé
Z	tué

# Les processus

---

- Options de ps:
  - a liste tous les processus actifs
  - u format d'affichage long
  - x inclut les processus sans terminal
- Tuer un processus:  
`kill -9 <PID>`
- Processus en arrière-plan: `&`  
(le terminal n'est pas bloqué)  
exemple: `emacs monfichier.c &`

# Les filtres

---

- Filtres simples

<b>cat</b>	<ul style="list-style-type: none"><li>– affiche le contenu des fichiers passés en paramètres (par défaut, <code>stdin</code>)</li><li>– options <code>-b</code>, <code>-n</code>, <code>-v</code></li></ul>
<b>more</b>	<ul style="list-style-type: none"><li>– affiche page par page les fichiers passés en paramètres (par défaut, <code>stdin</code>)</li><li><code>h</code> pour avoir le détail des commandes</li></ul>
<b>tee</b>	<ul style="list-style-type: none"><li>– recopie l'entrée <code>std</code> sur la sortie standard <b>et</b> dans le fichier passé en paramètre</li><li>– option <code>-a</code></li></ul>

exemples:

```
cat fic1 fic2
```

```
ls | tee liste.fic
```

```
more enormous_file
```

```
cat -n toto | more
```



Voir le man !!

# Les filtres

---

## Plus de filtres...

### sort

- trie l'entrée ligne par ligne
- options: `-r` (inverse l'ordre de tri)  
`+n` (ignore les  $n$  1<sup>ers</sup> champs)
- ex: `ls | sort`  
`ls -l | sort +4`

### comm

- sélectionne les lignes entre deux fichiers
- syntaxe: `comm [-123] fic1 fic2`
  - 1 = lignes de fic1 ( $\notin$  fic2)
  - 2 = lignes de fic2 ( $\notin$  fic1)
  - 3 = lignes communes

# Les filtres

---

## uniq

- détruit les lignes consécutives dupliquées
- options: **-u** (affiche les lignes "uniques"),  
**-d** (affiche les lignes "dupliquées")
- ex:

```
uniq -u fic
```

```
uniq -d fic
```

## diff

- compare deux fichiers
- options: **-b** (ignorer les lignes vides)
- ex:

```
diff fic1 fic2
```

# Les filtres

---

## cut

– sélectionne uniquement certaines colonnes du fichier passé en paramètre

– options:

–f<liste> : liste des champs à garder

–c<liste> : liste des colonnes à garder

–d<char> : séparateur de champs

– ex:

• `cut -c-10 rep.txt`

1 tonton 0

2 tux 0077

3 vuja 013

• `cut -f1,2 -d" " rep.txt`

1 tonton

2 tux

3 vuja

rep.txt

1 tonton 0311333300

2 tux 0077885566

3 vuja 0133220011



# Les filtres

---

## tr

- recopie `stdin` sur `stdout` en substituant des caractères
- syntaxe: `tr [-c ds] [s1 [s2]]`
- options:
  - `c` (complément de `s1`)
  - `d` efface les car. de `s1`
  - `s` tte séquence dans `s1` est substituée par un car. unique dans `s2`
- ex:
  - `tr A-Z a-z < essai`  
remplace les majuscules par des minuscules
  - `tr A-Z a-z < essai | tr -sc a-z '\012'`  
remplace les majuscules par des minuscules, puis remplace tout ce qui n'est pas une lettre minuscule par un retour chariot ('\012')

# Les filtres

---

## grep

- recherche, dans le fichier passé en paramètre, les lignes vérifiant une expression régulière donnée
- syntaxe : `grep expr_reg [fichier]`
- ex:
  - `grep 'toto' essai`  
cherche dans *essai* toutes les lignes qui contiennent le mot *toto*
  - `grep '^[A-Z]' essai`  
cherche dans *essai* toutes les lignes qui commencent par une majuscule
- (voir TP sur `grep` et les expressions régulières)

# Les filtres

---

- Et encore plein d'autres...  
sed, awk, cmp, ...

- Beaucoup de filtres et commandes...
- Savoir qu'elles existent
- Savoir ce qu'on peut en attendre
- Pour le reste, => **man !!**

