

Systemes d'exploitation

Valeur d'accueil et de reconversion en informatique (VARI1)

Daniel Porumbel (dp.cnam@gmail.com)

<http://cedric.cnam.fr/~porumbed/vari1/>

La vie sans système d'exploitation

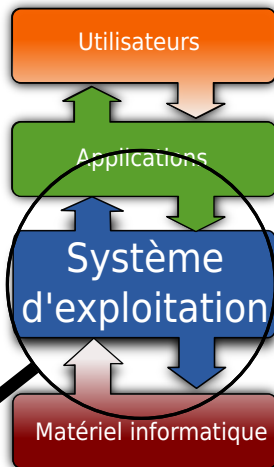
- Au tout début, les machines **ne possédaient pas** de système d'exploitation.
- Pour les faire fonctionner, il fallait que les programmes utilisateurs gèrent directement le CPU, la mémoire et les périphériques ce qui impliquait une **connaissance fine** des caractéristiques physiques et des spécificités des machines.
 - Un programme écrit sur une machine ne pouvait pas tourner sur une autre
- Pour libérer les programmeurs de la gestion du matériel, on a introduit la couche logicielle **Système d'Exploitation** sous la forme d'un ensemble de sous-programmes (appels système) standard.

Rappels couches génériques d'ordinateurs

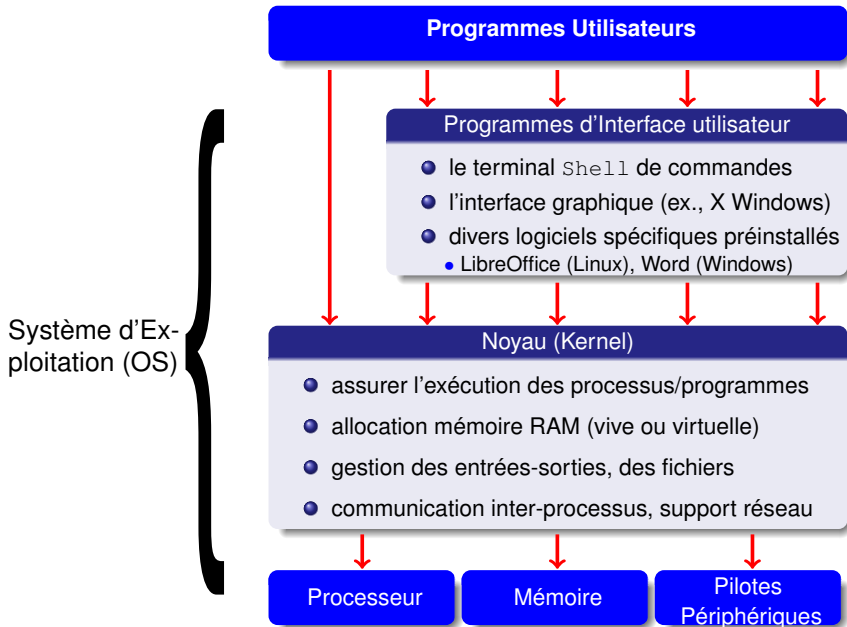
Couche Système d'Exploitation

- gère l'utilisation des ressources de la machine par les applications ;
- gestion programmes (multi-tache), communication inter-processus ;
- mémoire, le système de fichiers ;
- interface utilisateur, terminal shell et programmes utilitaires ;
- pilotes (en. : drivers) périphériques

Nous allons zoomer sur cette couche



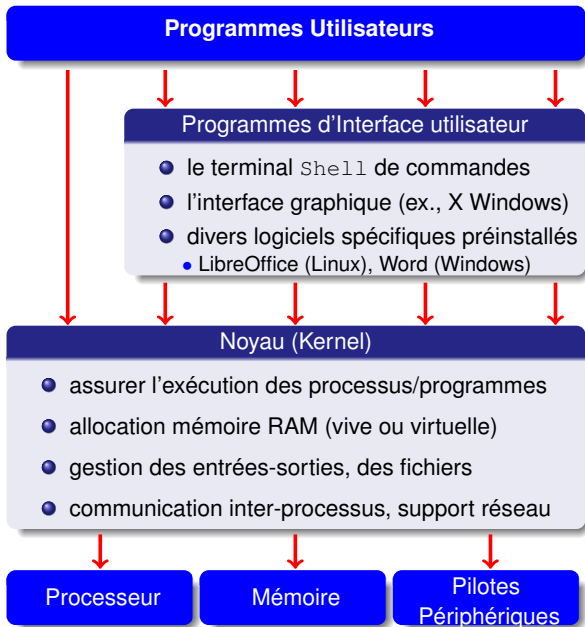
Composition Système d'Exploitation (OS)



Composition Système d'Exploitation (OS)

Pour lancer un programme (ex, Processing, navigateur Web, ...), il est d'abord chargé en mémoire et l'OS exécute les instructions

- Appels système aux fonctions de l'interface utilisateur, ex. la fonction `ellipse(...)` est envoyée à un serveur d'affichage
- Appels système au noyau (le premier programme chargé par l'OS pour gérer les fonctionnalités de base)



Les OS les plus populaires

Systèmes fermés «boites noires» : Windows, macOS, iOS, une partie d'Android

- beaucoup d'aspects sont secrets, fonctionnalités cachés(?)
 - ils peuvent **imposer leurs applications**, car il n'est pas toujours comode de les remplacer (logique de fermeture ?)
 - Peut-on installer l'application RATP d'Android sans se connecter à un compte Google ? Pour qoui ?
 - parfois compatibles **uniquement** avec des périphériques vendus par **la même entreprise** (Apple ?)
 - pas de support/applis pour les vieilles versions (pour vieux PC)
-
- Le macOS Darwin (open source) a été abandonné
 - Android est open source mais Google Play (le gestionnaire d'applis standard) ne l'est pas \implies Google reste encore et toujours maître des lieux sur les installations standard

Les OS les plus populaires

Systèmes fermés «boites noires» : Windows, macOS, iOS, une partie d'Android

- beaucoup d'aspects sont secrets, fonctionnalités cachés(?)
 - ils peuvent **imposer leurs applications**, car il n'est pas toujours comode de les remplacer (logique de fermeture ?)
 - Peut-on installer l'application RATP d'Android sans se connecter à un compte Google ? Pour qoui ?
 - parfois compatibles **uniquement** avec des périphériques vendus par **la même entreprise** (Apple ?)
 - pas de support/applis pour les vieilles versions (pour vieux PC)
-
- Le macOS Darwin (open source) a été abandonné
 - Android est open source mais Google Play (le gestionnaire d'applis standard) ne l'est pas \implies Google reste encore et toujours maître des lieux sur les installations standard

Les OS les plus populaires 2

Systèmes ouverts de type Unix/Linux


- logiciels libres et gratuits, **tout** est public (open source)
- forte **modularité**, ex., on peut garder un même OS mais changer l'interface graphique (passer de Gnome à KDE)
 - Un OS Linux est souvent très **configurable**, avec la liberté de tout modifier/échanger/personnaliser
 - Philosophie générale : l'OS n'est pas un « plat cuisiné » à ne pas modifier/toucher, vive la gastronomie !
- **macOs/iOs**, **Android** et **ChromeOS** utilisent du code d'un **noyau de type Unix** (BSD resp. Linux)

Il est possible de tourner un OS dans un autre à l'aide de logiciels de virtualisation, ex., virtualbox, voir osboxes.org.

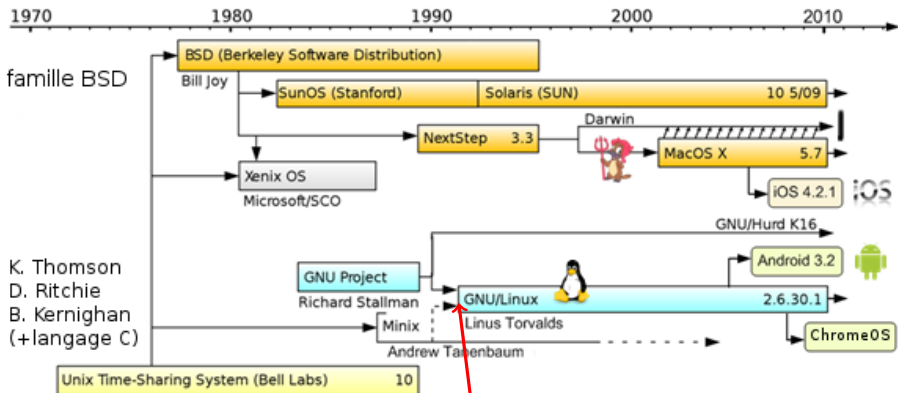
Les OS les plus populaires 2

Systèmes ouverts de type Unix/Linux

- logiciels libres et gratuits, **tout** est public (open source)
- forte **modularité**, ex., on peut garder un même OS mais changer l'interface graphique (passer de Gnome à KDE)
 - Un OS Linux est souvent très **configurable**, avec la liberté de tout modifier/échanger/personnaliser
 - Philosophie générale : l'OS n'est pas un « plat cuisiné » à ne pas modifier/toucher, vive la gastronomie !
- **macOs/iOs**, **Android** et **ChromeOS** utilisent du code d'un **noyau de type Unix** (BSD resp. Linux)

 Il est possible de tourner un OS dans un autre à l'aide de logiciels de virtualisation, ex., virtualbox, voir osboxes.org.

Histoire simplifié des OS/noyaux de type Unix/Linux



Le noyau Linux

- créé par Linus Torvalds en 1991 et initialement développé que par ce dernier, et ensuite par son équipe et des contributeurs partout autour du monde (c'est open-source !)
- Il est très petit (quelques MB) et peut tourner sur des machines plus anciennes

Interactions avec l'utilisateur

Tout système (OS) propose des fonctionnalités pour manipuler les fichiers/dossiers et pour lancer des programmes

- MacOS, Windows : logique de base “tout à la souris”
 - double clic sur des icônes, drag-and-drop, etc.
 - clic sur des menus déroulants
- Android (Google), iOS (Apple) : logique “écran tactile”

Dans ces cas, l'interface graphique masque le fonctionnement de l'OS, et peut rendre l'utilisateur incapable de résoudre des problèmes qui y sont liés

-
- Linux/Unix : à l'origine tout au clavier
 - Un `Shell` se présente sous la forme d'une interface en ligne de commande accessible depuis la console ou un terminal.
 - Plus tard : de nombreuses interfaces graphiques développées par les communautés “open source” (libres)
 - Toutes les couches et composants Linux/Unix peuvent être étudiées séparément : c'est le plus modulaire et transparent OS

⇒ Nous allons continuer notre étude avec Linux

Interactions avec l'utilisateur

Tout système (OS) propose des fonctionnalités pour manipuler les fichiers/dossiers et pour lancer des programmes

- MacOS, Windows : logique de base “tout à la souris”
 - double clic sur des icônes, drag-and-drop, etc.
 - clic sur des menus déroulants
- Android (Google), iOS (Apple) : logique “écran tactile”

-
- Linux/Unix : à l'origine tout au clavier
 - Un `Shell` se présente sous la forme d'une interface en ligne de commande accessible depuis la console ou un terminal.
 - Plus tard : de nombreuses interfaces graphiques développées par les communautés “open source” (libres)
 - Toutes les couches et composants Linux/Unix peuvent être **étudiées séparément** : c'est **le plus modulaire et transparent** OS

⇒ Nous allons continuer notre étude avec Linux

Linux : shell terminal et interface graphique

Le Shell Unix en ligne de commande

- Shell = enveloppe extérieure en anglais. On l'appelle Shell parce que c'est une couche autour du noyau.
- Il cache les détails de l'OS et gère les détails techniques des interactions avec le noyau
- Le premier shell est le *Thompson shell* apparu en 1971

Interfaces graphiques

- De nombreuses versions Open Source (ex., Gnome, KDE)
- Un *gestionnaire de fenêtres* se charge de l'affichage/placement des fenêtres (ex., compiz, marco, kWin, IceWm)
- Un serveur X reçoit les commandes de l'interface graphique et du gestionnaire de fenêtres
 - La commande `ssh -X IPMACHINE` permet de lancer sur l'écran local des programmes qui tournent sur IPMACHINE

Linux : shell terminal et interface graphique

Le Shell Unix en ligne de commande

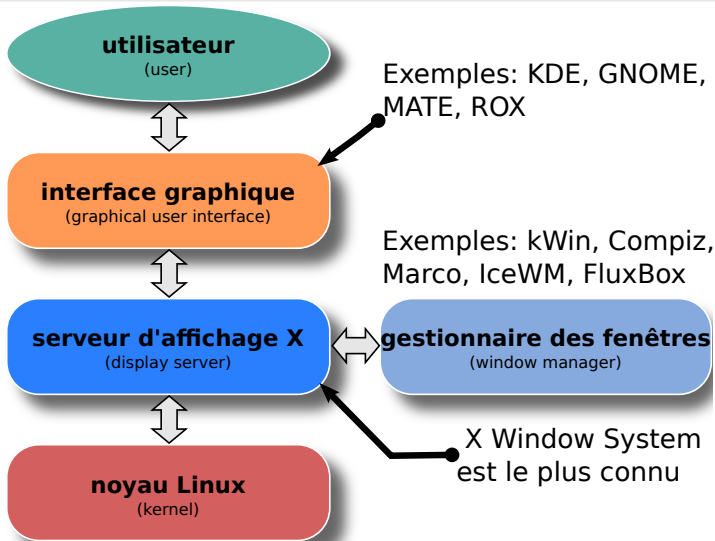
- Shell = enveloppe extérieure en anglais. On l'appelle Shell parce que c'est une couche autour du noyau.
- Il cache les détails de l'OS et gère les détails techniques des interactions avec le noyau
- Le premier shell est le *Thompson shell* apparu en 1971

Interfaces graphiques

- De nombreuses versions Open Source (ex., Gnome, KDE)
- Un *gestionnaire de fenêtres* se charge de l'affichage/placement des fenêtres (ex., compiz, marco, kWin, IceWm)
- Un serveur X reçoit les commandes de l'interface graphique et du gestionnaire de fenêtres
 - La commande `ssh -X IPMACHINE` permet de lancer sur l'écran local des programmes qui tournent sur IPMACHINE

Les couches graphiques de Linux

Image due à fr.wikipedia.org/wiki/IceWM



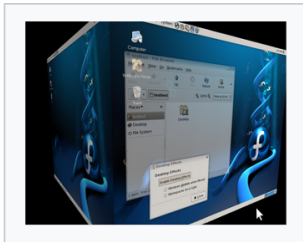
Tout est configurable ! Toutes les combinaisons sont possibles :
KDE-Compiz, Gnome-IceWM, Mate-Marco, Rox-IceWM,...

Exemples effets graphiques : Le gestionnaire fenêtres Compiz

- Spectaculaire mais consommation importante de ressources



l'effet « flammes » l'effet « lampe magique »



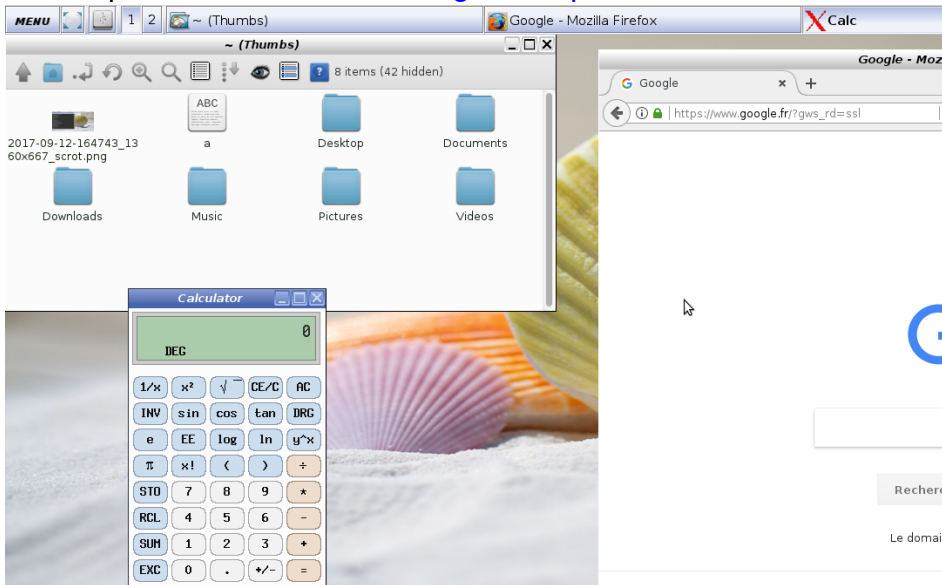
bureau sur un cube.



effet fenêtre molle.

Le gestionnaire de fenêtres IceWM

- Pas spectaculaire mais très léger et rapide

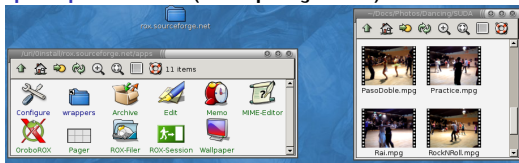


Exemple de systèmes légers/rapides

distribution : **Antix** (très complet)

noyau : Linux

interface graphique **Rox** (remplaçable)



gestionnaire fenêtres **IceWm** (remplaçable)

⇒ Ne pas jeter votre vieux ordinateur

TinyCoreLinux est encore plus petit et il y en a d'autres...

Facile à lancer dans `virtualbox` pour tester

La mémoire physique sous Linux

- les applications sont prioritaires pour l'utiliser
 - le noyau protège contre les accès illégaux des applications entre elles et vis à vis du noyau : impossible d'écrire sur la mémoire d'un autre programme ou dans le *kernel space*
- Mémoire Tampon (Buffers) et Cache : stocker dans la RAM une part du disque dur pour accélérer la machine
- SWAP : un fichier d'échange dit « swap » est utilisé lorsque la RAM est insuffisante (mémoire virtuelle)
 - Linux : faire une partition SWAP de même taille que la RAM

Disque(s) dur(s)

- \pm 50 fois moins rapides que la RAM
- Racine unique « / », pas de lecteur « C : » Windows
- Les clés USB, les DVD sont représentés par un fichier comme `/dev/sdb`, `/dev/sdc` qui doit être monté sur un dossier du système des fichiers

La mémoire physique sous Linux

- les applications sont prioritaires pour l'utiliser
 - le noyau protège contre les accès illégaux des applications entre elles et vis à vis du noyau : impossible d'écrire sur la mémoire d'un autre programme ou dans le *kernel space*
- Mémoire Tampon (Buffers) et Cache : stocker dans la RAM une part du disque dur pour accélérer la machine
- SWAP : un fichier d'échange dit « swap » est utilisé lorsque la RAM est insuffisante (mémoire virtuelle)
 - Linux : faire une partition SWAP de même taille que la RAM

Disque(s) dur(s)

- \pm 50 fois moins rapides que la RAM
- Racine unique « / », pas de lecteur « C: » Windows
- Les clés USB, les DVD sont représentés par un fichier comme `/dev/sdb`, `/dev/sdc` qui doit être monté sur un dossier du système des fichiers

Visualiser l'état des mémoires à l'aide du Shell

Démarrer une console/terminal et taper les commandes indiquées à gauche (explication à droite en commentaire) :

```
free                #informations sur la RAM
free -h             #h = human readable
cat /proc/meminfo
top                 #taper M pour trier selon
                   #la consommation memoire
df -h              #informations disques durs
```

Pour visualiser les clés USB montées :

```
cat /proc/mounts
mount
```

```
daniel@dcnam ~ $ mount
/dev/sda1 on / type ext4 (rw,errors=remount-ro)
/dev/sdb1 on /media/daniel/Barcelon type vfat (rw,nosuid,nodev) clé USB
```

Gérer d'autres ressources à l'aide du Shell

```
cat /proc/cpuinfo    #Informations CPU
cat /proc/version
uname --help         #affiche les options =>
uname -srv           #Kernel name, version, OS
```

Gérer un processus (ex. firefox) s'il consomme trop de CPU

```
killall -SIGSTOP firefox #firefox suspendu
...
killall -SIGCONT firefox #firefox redémarre
cpulimit -e firefox -l 50 #utiliser max 50% CPU
```

Arrêter un processus de manière forcé :

```
sudo killall -SIGKILL firefox
```

↑
sudo permet de passer en mode super-utilisateur (root), mot de passe demandé

Gérer d'autres ressources à l'aide du Shell

```
cat /proc/cpuinfo    #Informations CPU
cat /proc/version
uname --help         #affiche les options =>
uname -srv           #Kernel name, version, OS
```

Gérer un processus (ex. firefox) s'il consomme trop de CPU

```
killall -SIGSTOP firefox #firefox suspendu
...
killall -SIGCONT firefox #firefox redémarre
cpulimit -e firefox -l 50 #utiliser max 50% CPU
```

Arrêter un processus de manière forcé :

```
sudo killall -SIGKILL firefox
```

↑
sudo permet de passer en mode super-utilisateur (root), mot de passe demandé

Notion de dossier courant

- Comme tout processus Linux, le `shell` a un dossier courant.
- C'est à partir de ce dossier que ce fait le nommage des fichiers
- On peut l'afficher avec la commande `pwd`

Règles de nommage :

« `.` » = dossier courant

« `..` » = dossier parent/père

« `/` » = dossier racine du système de fichiers

« `~` » = dossier personnel (`$HOME`)

Commandes qui manipulent le dossier courant

- `cd /` se placer à la racine
- `cd /usr/games` aller au dossier «/usr/games»
- `cd ..` se placer dans le dossier parent
- `cd games` aller au dossier «games» du dossier courant
- `./fortune` lancer le programme « fortune » du dossier courant
- `cd ~` aller au dossier personnel (\$HOME)

Règles de nommage :

« . » = dossier courant

« .. » = dossier parent/père

« / » = dossier racine du système de fichiers

« ~ » = dossier personnel (\$HOME)

Shell : gestion de dossiers/fichiers

```
daniel@dcnam ~ $ cd /
daniel@dcnam / $ cd /usr/games/
daniel@dcnam /usr/games $ cd ..
daniel@dcnam /usr $ cd games
daniel@dcnam /usr/games $ ./fortune
Be careful of reading health books, you might die of a misprint.
    -- Mark Twain
daniel@dcnam /usr/games $ cd ~
daniel@dcnam ~ $ pwd
/home/daniel
daniel@dcnam ~ $
```

Règles de nommage :

- « . » = dossier courant
- « .. » = dossier parent/père
- « / » = dossier racine du système de fichiers
- « ~ » = dossier personnel (\$HOME)

Quelques commandes usuelles

```
Terminal
daniel@dcnam ~ $ cd /tmp/
daniel@dcnam /tmp $ mkdir vari1
daniel@dcnam /tmp $ cd vari1
daniel@dcnam /tmp/vari1 $ echo "Salut"
Salut
daniel@dcnam /tmp/vari1 $ echo "Salut">test.txt
daniel@dcnam /tmp/vari1 $ cat test.txt
Salut
daniel@dcnam /tmp/vari1 $ cp test.txt test2.txt
daniel@dcnam /tmp/vari1 $ ls
test2.txt test.txt
daniel@dcnam /tmp/vari1 $ mv test.txt test3.txt
daniel@dcnam /tmp/vari1 $ ls
test2.txt test3.txt
daniel@dcnam /tmp/vari1 $ rm test3.txt
daniel@dcnam /tmp/vari1 $ ls
test2.txt
daniel@dcnam /tmp/vari1 $ ls -l
total 4 propriétaire
-rw-r--r-- 1 daniel daniel 6 Sep 29 16:21 test2.txt
daniel@dcnam /tmp/vari1 $ cd ..
daniel@dcnam /tmp $ rm -r vari1
```

aller dans le dossier "/tmp/"
créer le dossier "vari1"
aller dans le dossier "vari1"
afficher un message
écrire "Salut" dans un fichier
afficher le fichier
copier test.txt en test2.txt
lister fichiers dossier courant
déplacer/renommer fichiers
effacer le fichier test3.txt
lister fichiers format long
taille fichier (octets)

Le manuel

Il est disponible si on tape `man` suivie de la commande qu'on veut étudier

- `man ls` : le manuel de la commande `ls` (lister fichiers)
- `man cat` : le manuel de la commande `cat` (afficher fichiers)

Pour naviguer dans la documentation :

- [ESPACE] : page suivante
- CTRL-u : page précédente
- / : chercher un mot clé
- n : aller à la prochaine apparition du mot clé

Redirection de la sortie standard

Fonctionnement classique :

- on tape une commande (ex, `ls`, `cat`);
- le résultat/sortie s'affiche dans **le terminal actif**

Il est possible de rediriger ce résultat :

`ls > fic.txt` → écrire le résultat dans le fichier `fic.txt` (écraser le contenu précédent)

`ls >> fic.txt` → ajouter le résultat au fichier `fic.txt` (sans rien écraser)

`ls -l /etc/ | more` → lancer `ls -l /etc/` et envoyer le résultat à la commande `more`, pour l'afficher page par page

Redirection de la sortie standard

Fonctionnement classique :

- on tape une commande (ex, `ls`, `cat`);
- le résultat/sortie s'affiche dans **le terminal actif**

Il est possible de rediriger ce résultat :

<code>ls > fic.txt</code>	→ écrire le résultat dans le fichier <code>fic.txt</code> (écraser le contenu précédent)
<code>ls >> fic.txt</code>	→ ajouter le résultat au fichier <code>fic.txt</code> (sans rien écraser)
<code>ls -l /etc/ more</code>	→ lancer <code>ls -l /etc/</code> et envoyer le résultat à la commande <code>more</code> , pour l'afficher page par page

En plus de la sortie, tout programme possède **une entrée** et **une sortie d'erreurs** généralement associées au terminal actif.

```
ls 2> err.txt → redirection des erreurs vers err.txt  
irb<<<"2+3" → la commande irb reçoit "2+3" comme entrée
```

Exemple : comparer les deux commandes suivantes

```
find / -name "virtual"  
find / -name "virtual" 2>/dev/null
```

- /dev/null=nulle part

En plus de la sortie, tout programme possède **une entrée** et **une sortie d'erreurs** généralement associées au terminal actif.

```
ls 2> err.txt → redirection des erreurs vers err.txt  
irb<<<"2+3" → la commande irb reçoit "2+3" comme entrée
```

Exemple : comparer les deux commandes suivantes

```
find / -name "virtual"  
find / -name "virtual" 2>/dev/null
```

- /dev/null=nulle part

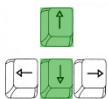
Touches très utiles : [Tab], flèches ↑↓, CTRL



Tapez le début d'une commande/fichier ⊕ appuyer sur la touche TAB : Linux **complète** la saisie ou propose les différentes possibilités pour la compléter

```
daniel@dcnam ~ $ echo "salut" > aadqlskdjqskskdjqsldk  
daniel@dcnam ~ $ cat aadqlskdjqskskdjqsldkjqsldjqsld
```

taper aa ↑ + [TAB] et linux remplit le reste!



Les flèches haut et bas permettent de revenir sur les commandes tapées récemment et de naviguer sur ces commandes

CTRL-C permet d'arrêter une commande (programme) lancée
CTRL-Z envoie la commande lancée en arrière-plan (taper `fg` pour revenir)

Un programme compilé est une “commande”

La syntaxe du lancement du programme exécutable est similaire à une commande :

exemple : `./monprogramme argument(s)`

`monprogramme` pourrait être compilé à partir de **C** ou **C++** ou **Pascal** ou tout autre langage

Un programme C tout petit : on donne la définition d'une fonction `main()` qui est exécutée en premier. Elle affiche «**Salut**» :

```
void main() {  
    printf("Salut");  
}
```

Compilation : `gcc main.c -o executb`

Exécution : `./executb`

gcc=compilateur C,
main.c=fichier C
(.c pas .cpp!)

Un programme compilé est une "commande"

La syntaxe du lancement du programme exécutable est similaire à une commande :

exemple : `./monprogramme argument(s)`

`monprogramme` pourrait être compilé à partir de **C** ou **C++** ou **Pascal** ou tout autre langage

Un programme C tout petit : on donne la définition d'une fonction **main()** qui est exécutée en premier. Elle affiche «**Salut**» :

```
void main() {  
    printf("Salut");  
}
```

Compilation : `gcc main.c -o executb`

Exécution : `./executb`

gcc=compilateur C,
main.c=fichier C
(.c pas .cpp!)

Un programme compilé est une "commande"

La syntaxe du lancement du programme exécutable est similaire à une commande :

exemple : `./monprogramme argument(s)`

`monprogramme` pourrait être compilé à partir de **C** ou **C++** ou **Pascal** ou tout autre langage

Un programme C tout petit : on donne la définition d'une fonction **main()** qui est exécutée en premier. Elle affiche «**Salut**» :

```
void main() {  
    printf("Salut");  
}
```

Compilation : `gcc main.c -o executb`

Exécution : `./executb`

gcc=compilateur C,
main.c=fichier C
(.c pas .cpp!)

Calculer le cube d'un argument en C

la fonction `main()` renvoie `void`=vide/rien

argument du programme/commande

```
void main(int argc, char** argv){
    int x, cube ;           //declaration variables
    x = atoi(argv[1]);     //conversion vers entier
    cube = x*x*x;
    printf("%d", cube);
}
```

- Tous les systèmes qu'on a vu sont écrits en C !
- Il est quasiment irremplaçable, il date des années 1970
- Il a influencé C++ et Java et par la suite Processing
⇒ Processing utilise une syntaxe de type C/C++

Calculer le cube d'un argument en C

la fonction `main()` renvoie `void`=vide/rien

argument du programme/commande

```
void main(int argc, char** argv){  
    int x, cube ; // declaration variables  
    x = atoi(argv[1]); // conversion vers entier  
    cube = x*x*x;  
    printf("%d", cube);  
}
```

- Tous les systèmes qu'on a vu sont écrits en C !
- Il est quasiment irremplaçable, il date des années 1970
- Il a influencé C++ et Java et par la suite Processing
⇒ Processing utilise une syntaxe de type C/C++