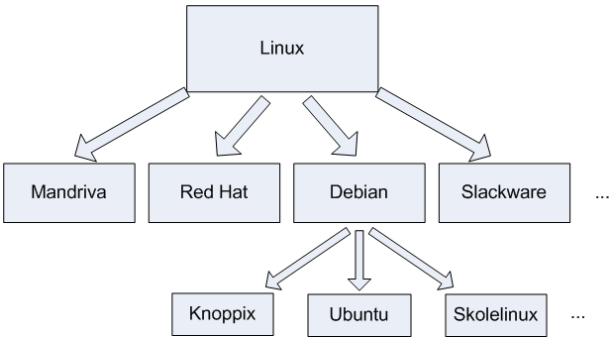
# Table of Contents

Linux & shell scripting ..... 2
La console ..... 3
Organisation des dossiers ..... 4
Manipuler les fichiers ..... 5
Les utilisateurs et les droits ..... 7
Nano, l'éditeur de texte du débutant ..... 9
Installer des programmes avec apt-get ..... 11
RTFM : Lisez le manuel ! ..... 12
Rechercher des fichiers ..... 14
Extraire, trier et filtrer des données ..... 16
Les flux de redirection ..... 19
Surveiller l'activité du système ..... 23
Exécuter des programmes en arrière-plan ..... 25
Exécuter un programme à une heure différée ..... 28
Archiver et compresser ..... 31
La connexion sécurisée à distance avec ssh ..... 33
Transférer des fichiers ..... 36
Analyser le réseau et filtrer le trafic avec un pare-feu ..... 38
Compiler un programme depuis les sources ..... 40
Vim : l'éditeur de texte du programmeur ..... 41
Introduction aux scripts shell ..... 45
Aficher et manipuler des variables ..... 46
Les conditions ..... 50
Les boucles ..... 52

# Linux & shell scripting

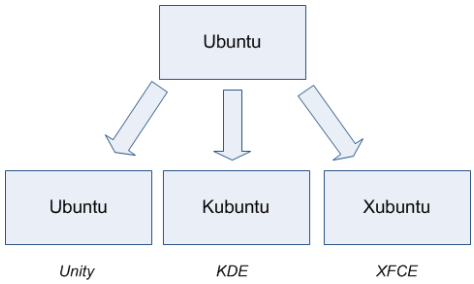
Le projet GNU - système d'exploitation libre et gratuit fonctionnant comme Unix lancé en 1984
Le projet GNU (programmes libres) et Linux (noyau d'OS) ont fusionné pour créer GNU/Linux.

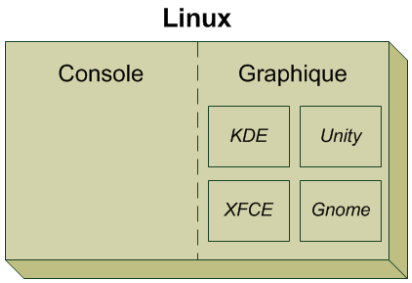
Les distributions de Linux : Slackware, Mandriva, RedHat, SuSE,
**Debian** (la seule distribution qui soit gérée par des développeurs indépendants plutôt que par une entreprise).
Les distributions de Debian : Knoppix, Skelelinux, **Ubuntu**




Ubuntu, la distribution la plus populaire:
- prévue pour le grand public; mises à jour fréquentes; beaucoup d'utilisateurs = beaucoup d'aide

Les gestionnaires de bureau (gérer les fenêtres, leur apparence, leurs options, etc.) de Ubuntu :
- **Unity** (par default), Gnome, KDE, XFCE



# La console

Passer en mode console : **Ctrl + Alt + F**{1, 2, 3, 4, 5, 6}
Retour au mode graphique : **Ctrl + Alt + F7**
Redémarrer le serveur X (graphique) : Ctrl + Alt + Backspace; remplacé par **Alt + PrintScreen + K**

**Console en mode graphique** : "terminal"
Lors du lancement de la console : on travaille par default dans le dossier personnel (home) ("~")
Le niveau d'autorisation: "$" = utilisateur normal, droits limités; "#" = superutilisateur, droits de root.

**Commandes :**
- date : affiche la date
- **ls** : équivalent à "list" = lister les fichiers et dossiers du répertoire courant

**Paramètres :**
- courts (une seule lettre, précédée par un tirer) : commande **-**d
  plusieurs : commande -d -a -U -h == commande -daUh
- longs (plusieurs lettres, précédées par deux tirets): commande **--**parametre

**Ex:** ls --all == ls -a (affiche tout le contenu du dossier, même les fichiers cachés (précédés par un point) )

**Valeurs des paramètres :**
- avec un paramètre court : commande **-**p 14
- avec un paramètre long : commande **--**parametre**=**14

**Autocomplétion de commande :**
- tapez les premières lettres, ensuite tapez 2x sur **Tab** pour afficher la liste des commandes possibles

Affichage **page par page** d'une requête :
- **Espace** = passer à la page suivante; **Entrée** = passer à la ligne suivante; **q** = arrêter la liste

**Historique** des commandes :
- naviguer dans la liste des commandes exécutées avec les flèches directionnelles haut et bas
- commande "history" pour afficher à l'écran toutes les commandes exécutées dans ce terminal
- recherche d'une commande tapée avec quelques lettres : **Ctrl + R**

**Raccourcis clavier pratiques :**
- **Ctrl + L** = commande "clear" = clear screen
- **Ctrl + D** = envoie le message "EOF" à la console; si la commande est vide cela fermera la console (exit)
- **Ctrl + C** = arrête l'exécution de la plupart des programmes console
- **Shift + PgUp** = remonter dans les messages envoyés par la console
- **Shift + PgDown** = redescendre dans les messages envoyés par la console

Les **raccourcis** utiles lorsque vous êtes en train d'écrire une **longue commande** :
- **Ctrl + A** / **Origine** (touche flèche pointant à NO) = ramène le curseur au début de la commande
- **Ctrl + E** / **Fin** = ramène le curseur à la fin de la commande
- **Ctrl + U** = supprime tout ce qui se trouve à gauche du curseur; si celui-ci est situé à la fin de la ligne, la ligne sera donc totalement supprimée
- **Ctrl + K** = supprime tout ce qui se trouve à droite du curseur; si celui-ci est situé au début de la ligne, la ligne sera donc totalement supprimée
- **Ctrl + W** = supprime le premier mot situé à gauche du curseur
- **Ctrl + Y** = si vous avez supprimé du texte avec une des commandes Ctrl + U, Ctrl + K ou Ctrl + W, cette commande "collera" le texte que vous venez de supprimer (un peu comme un copy-paste)

## Organisation des dossiers

Sous Linux il y a une seule racine (dossier de base qui contient tous les autres dossiers et fichiers), le "/".

La liste des dossiers les plus courants que l'on retrouve à chaque fois à la racine de Linux :

- **bin** : contient des programmes (exécutables) susceptibles d'être utilisés par tous les utilisateurs
- **boot** : fichiers permettant le démarrage de Linux
- **dev** : fichiers contenant les périphériques (contient des sous-dossiers pour chaque périphérique)
- **home** : répertoires personnels des utilisateurs; chaque utilisateur possède son dossier personnel
- **lib** : dossier contenant les bibliothèques partagées (gén. des fichiers .so) utilisées par les programmes
- **media** : lorsqu'un périphérique amovible (comme une carte mémoire SD ou une clé USB) est inséré dans votre ordinateur (monté), Linux vous permet d'y accéder à partir d'un sous-dossier de media
- **mnt** : c'est un peu pareil que media, mais pour un usage plus temporaire
- **opt** : répertoire utilisé pour les *add-ons* de programmes
- **proc** : contient des informations système
- **root** : c'est le dossier personnel de l'utilisateur « root »
- **sbin** : contient des programmes système importants
- **tmp** : dossier temporaire utilisé par les programmes pour stocker des fichiers
- **usr** : dossier dans lequel vont s'installer la plupart des programmes demandés par l'utilisateur
- **var** : ce dossier contient des données « variables », souvent des *logs*

Commandes utiles :
- **pwd** = *print working directory* = affiche le dossier actuel => /home/lorosanu
- **which** *programme* = **whereis** *programme* = afficher l'emplacement du *programme*
- **cd** *folder* = *change directory* = changer de répertoire courant vers "folder"
  (*folder* → représenté par un chemin relatif ou absolu; si absent => revenir au dossier personnel)
  (**cd ..** = revenir en arrière, au dossier "parent")
- **du** = *disk usage* = afficher la taille des dossiers (par default, la taille totale de chaque sous-dossier)
  ( du -h → la taille pour les humains; du -a → la taille des fichiers aussi; du -s → juste le total )
  ( du -h --max-depth=1 → afficher la tailles totale des dossiers courants)
- **df** = *disk free* = affiche la quantité d'espace disque disponible

La commande **ls** :

- -a → afficher tous les fichiers et dossiers cachés
- --color=auto → afficher des couleurs : dossiers en bleu, programmes en vert, le reste en blanc
- -F → afficher le type des éléments (dossiers suivis par un "/", programmes suivis par un "*", ...)
- -l → afficher la liste détaillée des éléments : droits, nombre de liens physiques, nom du propriétaire, groupe d'appartenance, taille (en octets), date de la dernière modification, nom; par default, l'ordre d'affichage est l'ordre alphabétique
- -lr → *r* = *reverse* => afficher les éléments en ordre alphabétique inverse
- -lh → *h* = *human readable* => afficher les tailles des éléments en Ko, Mo, Go, ...
- -lt → trier les éléments par date de dernière modification (default: tri par ordre alphabétique)

La commande `ls -larth` = afficher tous les éléments (même ceux cachés) en format détaillé et *human readable*, triés par ordre décroissante de la date de la dernière modification.

Commande utilisée fréquemment => créer un alias

```
alias ll  ls -larth
```

# Manipuler les fichiers

## Afficher le contenu des fichiers

**cat** → afficher tout le contenu du fichier d'un coup (option "-n" : affiche les numéros de ligne)

**more** / **less** → afficher le contenu du fichier progressivement (page par page, ligne par ligne, ...)
(différence : *less* est plus récent, avec plus des fonctionnalités)

**head** / **tail** → afficher le début et la fin du fichier

Raccourcis clavier pour **less** :

- **Espace** : afficher la page suivante (un "écran") du fichier
- **b** / **PageUp** : retourne en arrière d'un écran
- **Entrée** / **FlécheBas**: afficher la ligne suivante du fichier
- **y** / **FlècheHaut** : retourne d'une ligne en arrière
- d : afficher les onze ligne suivantes (soit une moitié d'écran)
- u : retourne en arrière de onze lignes
- **q** : arrête la lecture du fichier
- **h** : affiche l'aide
- **=** : indique où vous êtes dans le fichier (numéro de lignes affichées et pourcentage)
- **/** : mode recherche; tapez **/** suivi du texte recherché et faites **Entrée**; expressions régulières acceptées
  - **n** : après avoir fait une recherche avec **/**, la touche **n** vous permet d'aller à la prochaine occurrence
  - **N** : après avoir fait une recherche avec **/**, la touche **N** vous permet d'aller à l'occurrence précédente

**Head** : head [options] <filename>

| Arguments | Descriptions | Exemples |
|---|---|---|
| | default: affiche les 10 premières lignes | |
| **-n** | le nombre des lignes à afficher | -n2 ; -n 30 |
| **-c** | le nombre d'octets à afficher | -c1 ; -c 20 (affiche les premiers c caractères) |

**Tail** : tail [options] <filename>

| Arguments | Descriptions | Exemples |
|---|---|---|
| | default: affiche les 10 dernières lignes | |
| **-n** | le nombre des lignes à afficher | -n2 ; -n 30 |
| **-c** | le nombre d'octets à afficher | -c1 ; -c 20 (affiche les derniers c caractères) |
| **-f** | suit les derniers changements fait au fichier | tail -f file.txt |

## Créer des fichiers et dossiers

**touch** → met à jour la date de la dernière modification d'un fichier; si le fichier n'existe pas, il le crée
→ prend en paramètre le nom du fichier (ou les noms des fichiers, l'un l'après l'autre) à créer

**mkdir** → créer un dossier
→ prend en paramètre le nom du dossier (ou les noms des dossiers, l'un l'après l'autre) à créer
→ paramètre **-p** : créer tous les dossiers intermédiaires (ex: mkdir -p animaux/vertebres/chat)

## Copier et déplacer des fichiers et dossiers

**cp** → copier un fichier ; Syntaxe : cp <fileref> <filecopy> ; cp <fileref> <newpath/> ; ...
→ copier un dossier ; Syntaxe : cp **-R** <dirref> <dircopy>
→ utiliser le joker (wildcard) ***** : cp *****.jpg mondosssier/

**mv** → déplacer un fichier ; Syntaxe : mv <fileref> <newpath/>
→ renommer un fichier ; Syntaxe : mv <fileref> <newfilename>
→ déplacer un dossier ; Syntaxe : mv <dirref/> <newdir/>
→ utiliser le joker ***** : mv *****.jpg mondosssier/

## Supprimer des fichiers et dossier

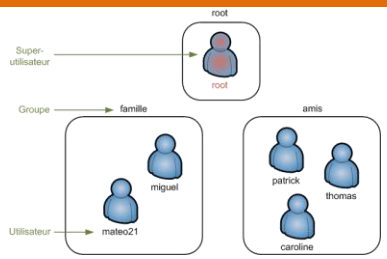
**rm** → supprimer un fichier ; Syntaxe : rm <filename> [<filename2> <filename3> ...]
→ paramètre **-i** : demander une confirmation (répondre par une lettre y / n, ensuite tapez Entrée)
→ paramètre **-f** : forcer la suppression, quoi qu'il arrive
→ paramètre **-v** (verbose) : demander à la commande de dire ce qu'elle est en train de faire
→ supprimer tout le contenu d'un dossier ; Syntaxe : rm **-r** <dirname> [<dirname2> <dirname3> ...]
→ utiliser le joker ***** : rm -rf *****.jpg (à utiliser avec précaution)

## Créer des liens entre les fichiers

**Lien physique** :
→ permet d'avoir deux noms de fichiers qui partages exactement le même contenu.
→ pointeur vers le contenu d'un fichier
→ Syntaxe : **ln** <filename1> <filename2>
→ faudra supprimer les deux fichiers pour supprimer le contenu.

**Lien symbolique** :
→ plus souvent utilisés pour faire des "raccourcis"
→ pointeur vers le nom d'un fichier / d'un dossier
→ Syntaxe : **ln -s** <filename1> <filename2>
→ si on supprime le fichier2, il ne se passe rien de mal
→ si on supprime le fichier1, fichier 2 pointera vers un fichier qui n'existe plus; le lien sera "mort"

# Les utilisateurs et les droits




## Sudo : exécuter une commande en root

Par défaut, vous êtes connectés sous votre compte limité.
Il est impossible sous Ubuntu de se connecter directement en root au démarrage de l'ordinateur.

On peut devenir root **temporairement** à l'aide de la commande **sudo** (**S**ubstitute **U**ser **DO**)

Exécuter une commande en root : **sudo** <commande>
Devenir root et le rester : **sudo su** (su = switch user) ; pour quitter le mode root : tapez "exit" ou Ctrl + D

## Gestion des utilisateurs

**adduser** → ajouter un utilisateur
Ex : adduser patrick [<groupname>] (par default, groupname= le nom de l'utilisateur)
→ ajout de l'utilisateur "patrick"
→ ajout du nouveau groupe "patrick"
→ ajout du nouvel utilisateur "patrick" avec le groupe "patrick"
→ création du répertoire personnel "/home/patrick/"
→ son compte et préconfiguré
→ faut lui associer un mot de passe

**passwd** → changer le mot de passe d'un utilisateur
Ex : passwd patrick
Attention : si vous ne précisez pas d'utilisateur, c'est le mot de passe de root que vous changerez

**deluser** → supprimer un compte d'utilisateur
Ex : deluser patrick
Ex : deluser --remove-home patrick (supprime aussi son home et tous ses fichiers personnels)
Attention : ne supprimez pas votre utilisateur; root ne doit pas se retrouver comme seul utilisateur sur la machine car ubuntu interdit de se logger en root

## Gestion des groupes

**addgroup** → créer un nouveau groupe (ex : addgroup amis)

**usermode** : modifier un utilisateur
paramètre -l : renomme l'utilisateur (le nom de son répertoire personnel ne sera pas changé)
paramètre -g : change de groupe
Ex : mettre patrick dans le groupe amis => usermod -g amis patrick
Ex : mettre patrick dans plusieurs groupes => usermod -G amis,collegues,paris patrick
Ex : ajouter un groupe dans la liste des groupes de patrick => usermod -aG collegues patrick

**delgroup** → supprimer un groupe

## Gestion des propriétaires d'un fichier

Seul l'utilisateur root peut changer le propriétaire d'un fichier.

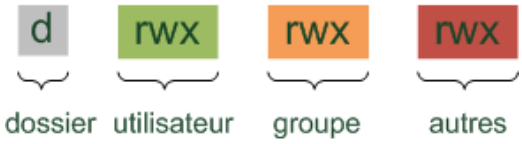
**chown** → changer l'utilisateur propriétaire d'un fichier
→ Syntaxe : chown <newuser> <filename>
→ Syntaxe : chown <newuser**:**newgroup> <filename> (change le groupe propriétaire au même temps)
→ Syntaxe : chown -R <newuser**:**newgroup> <dirname> (affecter récursivement les sous-dossiers)

**chgrp** → changer le groupe propriétaire d'un fichier
→ Syntaxe : chgrp <newgroup> <filename>

## Modifier les droits d'accès

Les droits d'accès du fichier ou dossier exprimés par cinq lettres différentes :

- **d** (directory) : indique si l'élément est un dossier
- **l** (link) : indique si l'élément est un lien (raccourci)
- **r** (read) : indique si on peut lire l'élément
- **w** (write) : indique si on peut modifier l'élément
- **x** (execute) : si c'est un fichier, « x » indique qu'on peut l'exécuter; ce n'est utile que pour les fichiers exécutables (programmes et scripts); si c'est un dossier, « x » indique qu'on peut le « traverser », c'est-à-dire qu'on peut voir les sous-dossiers qu'il contient
- **-** (tiret) : aucun droit




**chmod** → modifier les droits d'accès
Vous n'avez pas besoin d'être root pour utiliser chmod.
Vous devez juste être propriétaires du fichier dont vous voulez modifier les droits d'accès.

**Syntaxe**: **chmod** <droits> <filename> ; **chmod** -R <droits> <dirname>

Attribuer des droits avec des chiffres :
- chaque droit a attribué un chiffre : r = 4; w = 2; x = 1
- si vous voulez combiner ces droits, il va falloir additionner les chiffres correspondants

| **Droits** | --- | r-- | -w- | --x | rw- | -wr | r-x | rwx |
|---|---|---|---|---|---|---|---|---|
| **Chiffre** | 0 | 4 | 2 | 1 | 6 | 3 | 5 | 7 |
| **Calcul** | 0+0+0 | 4+0+0 | 0+2+0 | 0+0+1 | 4+2+0 | 0+2+1 | 4+0+1 | 4+2+1 |

Ex: "640" : indique les droits du propriétaire, du group et des autres (dans l'ordre)
Ex: "777" : indique le droit maximal que l'on puisse donner à tout le monde
Ex commande : **chmod** 600 rapport.txt

Attribuer des droits avec des lettres :
- u = user (propriétaire); g = group; o = other (autres)
- + = ajouter le droit; - = supprimer le droit; = = affecter le droit

Ex : **chmod** g+w rapport.txt (ajouter le droit d'écriture au groupe)
Ex : **chmod** o-r rapport.txt (enlever le droit de lecture aux autres)
Ex : **chmod** u+rx rapport.txt (ajouter les droits de lecture et exécution au propriétaire)
Ex : **chmod** g+w,o-w, **chmod** go-r, **chmod** +x, **chmod** u=rwx,g=r,o=-

# Nano, l'éditeur de texte du débutant

## Premiers pas avec nano

Pour démarrer le logiciel, il vous suffit simplement de taper **nano** dans la console.


**Raccourcis** les plus importants :

- **Ctrl + G** : afficher l'aide
- **Ctrl + K** : couper la ligne de texte
- Ctrl + U : coller la ligne de texte que vous venez de couper
- **Ctrl + C** : afficher à quel endroit du fichier votre curseur est positionné (numéro de ligne…)
- **Ctrl + W** : rechercher dans le fichier (tapez le mot → Entrée; retapez Ctrl+W pour aller à l'occ suivante)
- **Ctrl + O** : enregistrer le fichier (écrire)
- **Ctrl + X** : quitter Nano
- Escape X : enlever / réafficher l'aide mémoire

Vous pouvez vous déplacer dans le fichier avec les flèches du clavier,
ainsi qu'avec les touches PageUp et PageDown pour avancer de page en page.

**Paramètres** de la commande nano :

**-m** : autorise l'utilisation de la souris sous nano

**-i** : indentation automatique

**-A** : active le retour intelligent au début de la ligne

=> lorsqu'on appuie sur *Origine*, le curseur se positionne au début de la ligne, après les alinéas

=> lancement : **nano -miA** rapport.txt

## Configurer nano avec .nanorc

**.nanorc** = le fichier des configuration de nano pour l'utilisateur courant, lu par nano à chaque exécution

Exemples des **options**:

set mouse = équivalent à -m = nano sera automatiquement chargé avec la prise en charge de la souris

set autoindent = équivalent à -i

set smarthome = équivalent à -A

include "/usr/share/nano/c.nanorc" = activer la coloration "intelligente" de vos fichiers, selon leur type

Pour appliquer ces options pour tous les utilisateur de la machine courante, faudra modifier le fichier de configuration global de nano situé dans /etc/nanorc (par root).

## Configurer sa console avec .bashrc

Le fichier **.bashrc** vous permet entre autres choses de personnaliser l'invite de commandes.

Les alias = des commandes utilisateur qui sont automatiquement transformées en d'autres commandes

Syntaxe: alias nom=commande

alias ls='ls --color=auto'
alias ll='ls -lh'
alias la='ls -A'
alias l='ls -CF'
alias rm='rm --preserve-root'
alias cp='cp -i --preserve=mode,timestamps'

Le bashrc global : situé sous /etc/bash.bashrc

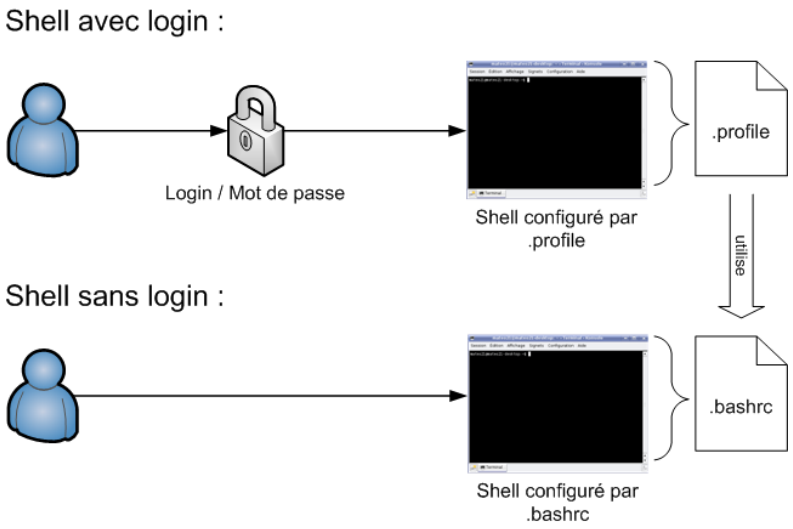
## Le .profile

De même qu'il existe un ~/.bashrc et un /etc/bash.bashrc, il existe un ~/.profile et un /etc/profile.

Le .profile est lu à chaque nouvelle console dans laquelle vous vous loggez.
C'est le cas des consoles que vous ouvrez avec Ctrl + Alt + F1 à F6 (tty1 à tty6).

Le .bashrc est lu lorsque vous ouvrez une console dans laquelle vous ne vous loggez pas.
C'est le cas des consoles que vous ouvrez en mode graphique (Terminal sous Unity, Konsole sous KDE).

Le .profile fait appel au .bashrc.



# Installer des programmes avec apt-get

## Les paquets et leurs dépendances

Un paquet est une sorte de dossier zippé qui contient tous les fichiers du programme.
Il se présente sous la forme d'un fichier.deb, en référence à **DEB**ian.
Il contient toutes les instructions nécessaires pour installer le programme.

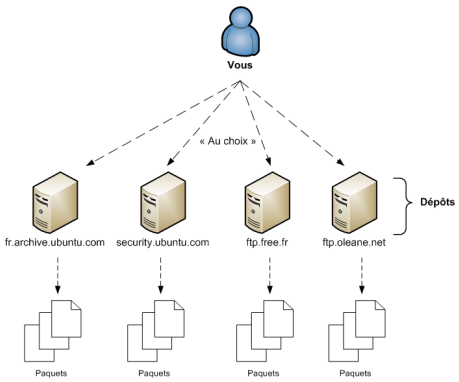
Différences par rapport aux exécutables Windows :

- il y a une gestion des **dépendances** du programme
  (les programmes dépendent d'autres programmes pour fonctionner)
- on n'a pas besoin de faire une recherche sur un moteur de recherche pour trouver un .deb;
  tous les .deb sont rassemblés au même endroit sur un même serveur appelé **dépôt** (*repository*)

Il existe en fait un grand nombre de dépôts.

La plupart proposent exactement les mêmes paquets (les dépôts sont donc des copies les uns des autres). Certains dépôts proposent toutefois des programmes que l'on ne trouve nulle part ailleurs, mais il est rare que l'on ait besoin de s'en servir.

C'est donc à vous de choisir le dépôt que vous voulez utiliser (chacun de ces dépôts est identique). Il est conseillé de choisir un serveur qui soit proche de chez vous (pour téléchargez suffisamment vite).




La liste des dépôts que votre ordinateur utilise est stockée sous /etc/apt/sources.list
Chaque ligne du fichier commence par une de ces deux directives :
- deb : pour télécharger la version compilée (binaire) des programmes (version « prête à l'emploi »)
- deb-src : permet de récupérer le code source du programme (c'est l'avantage des logiciels libres de pouvoir consulter la source des programmes; mais généralement vous en avez pas besoin)

**Interface graphique** : Système → Administration → Sources de logiciels

## Les outils de gestion des paquets

Les deux programmes console de gestion des paquets les plus connus sont : **apt-get** et **aptitude**

Nous devons généralement suivre trois étapes pour télécharger un paquet :
- **apt-get update** (optionnel) :
  → pour mettre notre cache à jour (télécharger la nouvelle liste des paquets proposés par le dépôt)
- **apt-cache search** *monpaquet* (optionnel) :
  → pour rechercher le paquet que nous voulons télécharger, lorsqu'on connait pas déjà le nom exact
- **apt-get install** *monpaquet* : pour télécharger et installer notre paquet

Pour désinstaller un paquet : **apt-get remove** *monpaquet*
Pour désinstaller également les dépendances qui deviendront inutiles : **apt-get autoremove** *monpaquet*

Pour mettre à jour tous les paquets d'un seul coup : **apt-get upgrade**.
Pensez à faire un apt-get update pour mettre à jour le cache des paquets avant de lancer un upgrade.

# RTFM : Lisez le manuel !

## man : afficher le manuel d'une commande

La commande man prend en paramètre le nom de la commande dont vous voulez lire la doc.

Ex: man mkdir

Se déplacer dans le manuel :

- les **touches fléchées** du clavier (vers le haut et vers le bas) pour vous déplacer ligne par ligne
- les touches **PageUp** et **PageDown** (ou **Espace**) pour vous déplacer de page en page
- la touche **Home** (aussi appelée Origine) pour revenir au début du manuel, et sur **Fin** pour aller à la fin
- la touche **/** pour effectuer une recherche ; tapez ensuite le mot que vous recherchez dans le manuel puis appuyez sur Entrée. Si la recherche renvoie un résultat vous serez automatiquement placés sur le premier résultat trouvé. Pour passer au résultat suivant, tapez à nouveau / puis directement sur Entrée (sans retaper votre recherche).
- la touche **q** pour quitter le manuel à tout moment

Les sections d'un manuel :

- NAME : le nom de la commande ainsi qu'une courte description de son utilité
- SYNOPSIS : c'est la liste de toutes les façons d'utiliser la commande
- DESCRIPTION : une description plus approfondie de ce que fait la commande. On y trouve aussi la liste des paramètres et leur signification
- AUTHOR : l'auteur du programme. Il y a parfois de nombreux auteurs
- REPORTING BUGS : si vous rencontrez un bug dans le logiciel, on vous donne l'adresse de la personne à contacter pour le rapporter.
- COPYRIGHT : le *copyright*, c'est-à-dire la licence d'utilisation de la commande. La plupart des programmes que vous utilisez sont certainement des programmes open source sous licence GPL, ce qui vous donne le droit de voir la source et de redistribuer le programme librement.
- SEE ALSO : cette section vous propose de « voir aussi » d'autres commandes en rapport avec celle que vous êtes en train de regarder. C'est une section parfois intéressante.

## Comprendre le synopsis

Le rôle de synopsis est de lister toutes les façons possibles d'utiliser la commande : toutes les combinaisons de paramètres que l'on peut réaliser avec cette commande.

**Ex :** le synopsis de **mkdir** : `mkdir [OPTION] DIRECTORY...`

Détaillons point par point ce SYNOPSIS.

- mkdir : pour utiliser la commande mkdir, vous devez commencer par taper mkdir

- [OPTION] : après mkdir, vous pouvez écrire une option; les crochets indiquent que c'est facultatif

- DIRECTORY : le nom du répertoire à créer. Ce paramètre est obligatoire (il n'est pas entre crochets)

- **...** : le terme DIRECTORY est suivi de points de suspension. Cela signifie que l'on peut répéter DIRECTORY autant de fois que l'on veut (=> on peut indiquer plusieurs répertoires à la fois)

Les mots du SYNOPSIS écrits **en gras** sont des mots à taper tels quels.
Les mots **soulignés**, eux, doivent être remplacés par le nom approprié.

Les options facultatives sont listées dans la section `DESCRIPTION` du man.

**Ex :** le synopsis de **cp** :

```
cp [OPTION]... [-T] SOURCE DEST
cp [OPTION]... SOURCE... DIRECTORY
cp [OPTION]... -t DIRECTORY SOURCE...
```

- La première ligne : `cp [OPTION]… [-T] SOURCE DEST`
  Les paramètres obligatoires : `SOURCE` (nom du fichier à copier) et `DEST` (nom de la copie).
  Ces fichiers peuvent être précédés d'une ou plusieurs options, ainsi que de l'option `-T`.
- La seconde ligne est un peu différente : `cp [OPTION]… SOURCE… DIRECTORY`
  Cette fois, on peut copier un ou plusieurs fichiers (`SOURCE…`) vers un répertoire (`DIRECTORY`).
  Tout cela peut encore une fois être précédé d'une ou plusieurs options.
- Enfin, la troisième ligne : `cp [OPTION]… -t DIRECTORY SOURCE…`
  Signifie qu'on peut aussi écrire le répertoire (`DIRECTORY`) dans un premier temps, suivi d'un ou plusieurs fichiers (`SOURCE…`). Attention, vous remarquez que dans ce cas il est obligatoire d'utiliser le paramètre `-t` qui n'est plus entre crochets.

Exemples d'utilisation :

1: **cp** photo.jpg photo_copie.jpg

1: **cp** -vi photo.jpg photo_copie.jpg

2: **cp** photo.jpg photo_copie.jpg images/

## Trouver une commande : apropos

La commande **apropos** est un peu l'inverse de man : elle permet de retrouver une commande.
Paramètre : un mot clé qu'elle va rechercher dans les descriptions de toutes les pages du manuel.

Prenons un exemple :
- une commande (que vous avez installée) en rapport avec le son pour modifier le volume en console
- vous pouvez taper : **apropos** sound

… ce qui va rechercher toutes les commandes qui parlent de son (sound) dans leur page du manuel.
=>

```
alsactl (1)      - advanced controls for ALSA soundcard driver
alsamixer (1)    - soundcard mixer for ALSA soundcard driver, with ncurse...
amixer (1)       - command-line mixer for ALSA soundcard driver
```

À gauche la commande, à droite l'extrait de sa description contenant le mot recherché
Il se trouve que ce que je cherchais était alsamixer.

## D'autres façons pour lire le manuel

Le paramètre -h (et --help) présent dans la plupart des commandes.

Ce paramètre provoque l'affichage d'une aide résumée.
Parfois cette aide est d'ailleurs plus facile à lire que celle du man.
Ex: apt-get -h

La commande **whatis** :
- est une sorte de man tres allégé
- elle donne juste l'en-tête du manuel pour expliquer en deux mots à quoi sert la commande.

Ex: **whatis** mkdir

# Rechercher des fichiers

## locate : une recherche rapide

Son utilisation est intuitive, il suffit d'indiquer le nom du fichier que vous voulez retrouver.

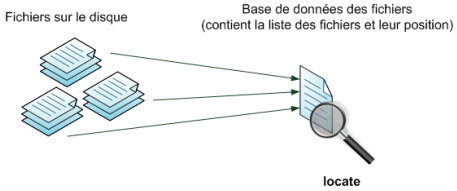
Ex: **locate** notes.txt => /home/mateo21/notes.txt

Ex: **locate** australie

=>
/home/mateo21/photos/australie1.jpg
/home/mateo21/photos/australie2.jpg
/home/mateo21/photos/australie3.jpg

**locate** vous donne tous les fichiers qui contiennent le mot « australie » dans leur nom.
Que ce soient des fichiers ou des dossiers, elle ne fait pas la différence.
Elle vous donne la liste complète des fichiers qu'elle a trouvés.

La commande ne fait pas la recherche sur votre disque dur entier,
mais seulement sur une base de données de vos fichiers (qui est mise à jour une fois par jour).

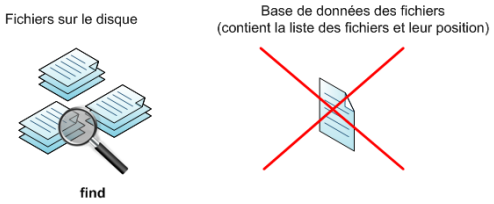



Vous pouvez forcer la commande **locate** à reconstruire la base de données des fichiers du disque dur.
Cela se fait avec la commande **updatedb**, à exécuter en root (avec sudo) : sudo updatedb

## find : une recherche approfondie

**Find** est la commande de recherche par excellence pour retrouver des fichiers,
mais aussi pour effectuer des opérations sur chacun des fichiers trouvés.

**Find** va parcourir tout le disque dur pour rechercher des fichiers.




La commande **find** s'utilise de la façon suivante : **find** [<where>] <what> [<do>]

- **<where>** : nom du dossier dans lequel va se faire la recherche. Tous les sous-dossiers seront analysés. Il est donc possible de limiter la recherche à `/home` par exemple. En abscence de ce paramètre, la recherche s'effectuera dans le dossier courant et ses sous-dossiers.
- **<what>** : c'est le fichier à rechercher. On peut rechercher un fichier par son nom, mais aussi en fonction de la date de sa création, de sa taille, etc. Ce paramètre est **obligatoire**.
- **<do>** : il est possible d'effectuer des actions automatiquement sur chacun des fichiers trouvés (on parle de « post-traitement »). Par défaut, la commande `find` affiche les résultats trouvés et ne fait rien d'autre avec.

Utilisation basique de la commande find :

- recherche à partir du nom : **find -name** "logo.png"

  → find récupère uniquement la liste des fichiers qui s'appellent exactement comme demandé.
  → s'il existe un fichier nommé logo2.png, il ne sera pas listé dans les résultats
  → pour qu'il le soit, il faut utiliser le joker : l'étoile « * » !
  → pour effectuer la recherche sous tout le disque dur : **find / -name** "logo.png"

- recherche à partir de la taille: **find -size** +10M

  → +10M = fichiers de plus de 10 Mo; on peut aussi utiliser K pour Ko, G pour Go, ...
  → -10 M = fichiers de moins de 10 Mo
  → 10 M = fichiers de 10 Mo exactement (ni plus, ni moins)

- recherche à partir de la date de dernier accès: **find -name** "*.odt" **-atime** -7
  → -7 = depuis 7 jours

- recherche uniquement des répertoires ou des fichiers : **find -type** d / **find -type** f

Utilisation avancée avec manipulation des résultats :

- par défault, on affiche les noms des fichiers :

  **find** -name "*.png" == **find** -name "*.png" **-print**

- formater l'affichage :

  **find** -name "*.png" **-print** "%p - %u\n"  (%p = nom du fichier; %u = nom du proprietaire)

- supprimer les fichiers trouvés :

  **find** -name "*.png" **-delete**

  **find** . -type f -name '*~' **-delete**

- appeler une commande qui effectuera une action sur chacun des fichiers trouvés.:

  **find** -name "*.png" **-exec** chmod 600 {} \;

  La commande qui suit -exec :

  - ne doit PAS être entre guillemets
  - les accolades **{}** seront remplacées par le nom du fichier
  - la commande doit finir par un **\;** obligatoirement

  Vous pouvez utiliser `-ok` à la place de `-exec`.
  Le principe est le même, mais on vous demandera une confirmation pour chaque fichier rencontré.

# Extraire, trier et filtrer des données

## grep : filtrer des données

Rechercher un mot dans un fichier et afficher les lignes dans lesquelles ce mot a été trouvé.
Elle peut être utilisée de manière très simple ou plus complexe (avec des expressions régulières).

Utiliser grep simplement : **grep** [options] <text> <filename>

Le 1er paramètre est le texte à rechercher, le 2nd est le nom du fichier dans lequel est faite la recherche.

Paramètres :

**-i** = ignore case = ne tient pas compte de la case des lettres (majuscules ou minuscules)

**-n** = connaître les numéros des lignes retournées

**-w** = sélectionner les lignes correspondant à des mots entiers

**-v** = inverser la rechercher = afficher toutes les lignes qui ne contiennent pas le texte donné

**-o** = affiche seulement la chaine trouvée (à la place de la ligne entière)

**-c** = affiche seulement le nombre de lignes correspondant à la recherche

**--color** = les correspondances réussies seront mises en évidence

**-r** = rechercher dans tous les fichiers et sous-dossiers d'un dossier (la commande **rgrep** est équivalente)

Utiliser grep avec des regex : **grep -E** <regex> <filename> (la commande **egrep** est équivalente)

| Caractère spécial | Signification |
|---|---|
| . | caractère quelconque |
| ^ | début de ligne |
| $ | fin de ligne |
| [] | un des caractères entre les crochets |
| ? | l'élément précédent est optionnel (peut être présent 0 ou 1 fois) |
| * | l'élément précédent peut être présent 0, 1 ou plusieurs fois |
| + | l'élément précédent doit être présent 1 ou plusieurs fois |
| \| | ou |
| () | groupement d'expressions |

Exemples:

```
grep -E Alias    .bashrc   => # Alias definitions
grep -E ^Alias   .bashrc   => rien
grep -E ^alias   .bashrc   => alias ll='ls -lArth'
grep -E [Aa]lias .bashrc   => renvoie toutes les lignes qui contiennent « alias » ou « Alias »
grep -E [0-4]    .bashrc   => renvoie toutes les lignes qui contiennent un nombre compris entre 0 et 4.
grep -E [a-zA-Z] .bashrc   => renvoie toutes les lignes qui contiennent un caractère alphabétique
```

## sort : trier les lignes

La commande sort se révèle bien utile lorsqu'on a besoin de trier le contenu d'un fichier.

Syntaxe : **sort** [options] <filename>
Le contenu du fichier est trié alphabétiquement et le résultat est affiché dans la console.
Sort ne fait pas attention à la casse (majuscules / minuscules).

Paramètres :
**-n** = numbers = trier des nombres
**-r** = reverse = trier en ordre inverse
**-R** = random = trier aléatoirement
**-u** = unique = supprimer les doublons
**-f** = ignore case = ne tient pas compte de la case des lettres (majuscules ou minuscules)
**-i** = ignorer les caractères non imprimables
**-b** = blanks = ignorer les espaces au début des lignes
**-c** = check = vérifie seulement si le fichier est trié
**-o <file>** = output = écrire le résultat dans un fichier

**-k** <f1>,<f2> = considère seulement ces champs pour le tri
**-t** <del> <columns> = spécifie le délimiteur (tab par default) entre les colonnes de l'entrée

Exemples :

```
sort -n nombres.txt
sort -r noms.txt
sort -o noms_tries.txt noms.txt
sort -t'|' +2-4                  # => tri par la 3eme et 4eme colonne
sort -k 3.1,3.5                  # => tri par les 5 premiers caracteres du 3eme champ de chaque ligne
```

## wc : compter le nombre de lignes

La commande **wc** signifie word count.

C'est donc a priori un compteur de mots mais en fait, on lui trouve plusieurs autres utilités :
compter le nombre de lignes (très fréquent) et compter le nombre de caractères.

Syntaxe : wc [options] <filename>

Sans paramètre :

```
wc noms.txt          => 8  8 64 noms.txt
```

Ces résultats signifient dans l'ordre : 8 = nombre de lignes; 8 = nombre de mots; 64 = nombre d'octets

Paramètres :
**-l** = lines = nombre de lignes
**-w** = words = nombre de mots
**-c** = nombre d'octets
**-m** = nombre de caractères

## uniq : supprimer les doublons

La commande **uniq** est à comme objectif de détecter ou supprimer les lignes en double dans un fichier.
**Uniq** ne repère que les lignes successives qui sont identiques; il faut donc travailler sur un fichier **trié**.

Syntaxe : uniq [options] <filename>

Paramètres :
**-c** = count = comptez le nombre d'occurrences de chaque ligne
**-u** = unique = afficher seulement les lignes qui n'ont pas des doublons
**-d** = doubles = afficher seulement les lignes qui ont des doublons
**-D** = afficher les lignes qui ont des doublons (y compris les lignes dupliquées)
**-w** <x> = compare seulement les premiers w caractères (et non la ligne entière)
**-s** <x> = skip = ignore les premiers s caractères dans la comparaison
**-f** <x> = fields = ignore les premiers f champs dans la comparaison (le délimiteur par default et l'espace)

Exemples :

```
uniq noms_tries.txt        # affiche la liste de noms sans doublons
uniq -c -w 8               # affiche le nombre des lignes qui ont les mêmes premiers 8 caractères
uniq -d -s 2               # affiche les lignes dupliquées : comparaison faite à partir du 3eme caractère
```

## cut : couper une partie du fichier

**Cut** coupe du texte au sein d'un fichier afin de conserver uniquement une partie de chaque ligne.
== extraire des sections de chaque ligne d'un fichier

Syntaxe : **cut** [-b [-n] ] [-c list] [-f list [-d delim] ] [-s] <filename>

Paramètres :
**-c** <list> = characters = couper selon le nombre de caractères (nb d'octets; attention aux accents ...)
**-f** <list> = fields = couper selon les champs (champs définis par rapport au délimiteur)
**-d** <del> = changer le délimiteur par default (Tab)
**-s** = skip = ignore les lignes qui ne contiennent pas des délimiteurs
**-b** <list> = bytes = couper selon le nombre de bytes
(combiné avec "-n" : ne sépare pas les caractères de plusieurs octets)

Les paramètres -f et -d sont utilisés le plus souvent ensemble :
-d indique quel est le délimiteur des données dans le fichier
-f indique le numéro du ou des champs à couper.

Exemples :

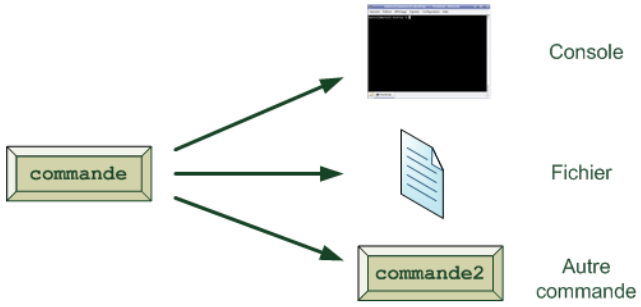
```
cut -c 2-5 noms.txt        # conserver uniquement les caractères 2 à 5 de chaque ligne du fichier
cut -c  -3 noms.txt        # conserver uniquement les 3 premiers caractères de chaque ligne du fichier
cut -c 3-  noms.txt        # conserver du 3eme au dernier caractère de chaque ligne du fichier
cut -d , -f 1   notes.txt  # conserver uniquement le 1er champ (les champs sont séparés par une virgule)
cut -d , -f 1,3 notes.txt  # conserver le 1er et 3eme champs (les champs sont séparés par une virgule)
cut -d , -f  -3 notes.txt  # conserver les 3 premiers champs (les champs sont séparés par une virgule)
```

## Les flux de redirection

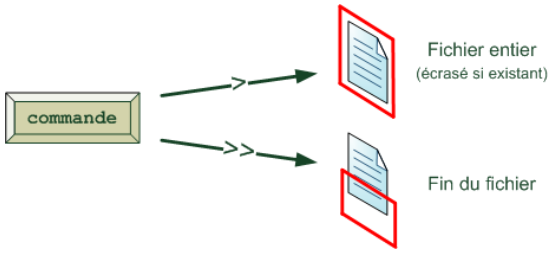
Le fonctionnement des commandes proposées dans la console de Linux :

- vous tapez une commande
- le résultat s'affiche dans la console (à l'ecran)

Il est neanmoins possible de **rediriger** ce résultat.
Au lieu que celui-ci s'affiche dans la console, vous allez pouvoir l'envoyer ailleurs :
dans un fichier ou en entrée d'une autre commande pour effectuer des « chaînes de commandes ».




### > et >> : rediriger le résultat dans un fichier




**>** (appelé chevron) : rediriger le résultat de la commande dans le fichier de votre choix *(write)*

```
cut -d , -f 1 notes.csv > eleves.txt          # => rien ne s'affichera dans la console
```

Tout aura été redirigé dans un fichier appelé "eleves.txt"
qui vient d'être créé pour l'occasion dans le dossier dans lequel vous vous trouviez.

Attention : si le fichier existait déjà il sera écrasé sans demande de confirmation !

Parfois, vous ne voulez ni voir le résultat d'une commande ni le stocker dans un fichier.
Dans ce cas, l'astuce consiste à rediriger le résultat dans **/dev/null**.
C'est un peu le « trou noir » de Linux : tout ce qui va là-dedans disparaît immédiatement.

Par exemple : `commande_bavarde > /dev/null`

**>>** (double chevron) : rediriger le résultat de la commande à la fin du fichier de votre choix *(append)*

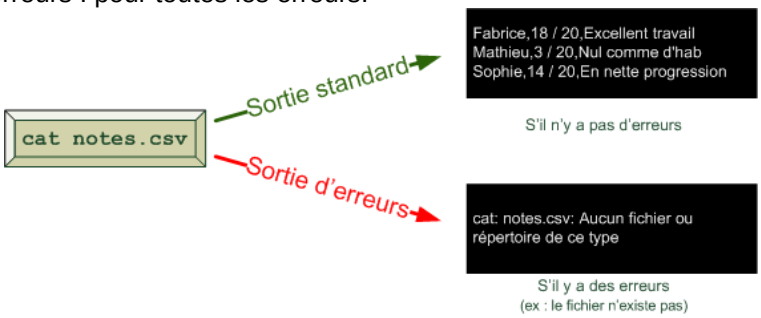
```
cut -d , -f 1 notes.csv >> eleves.txt          # => rien ne s'affichera dans la console
```

Avantages :

- vous ne risquez pas d'écraser le fichier s'il existe déjà
- si le fichier n'existe pas, il sera créé automatiquement

## 2>, 2>> et 2>&1 : rediriger les erreurs

Toutes les commandes produisent deux flux de données différents :
- la sortie standard : pour tous les messages (sauf les erreurs) ;
- la sortie d'erreurs : pour toutes les erreurs.




Par défaut, tout s'affiche dans la console : la sortie standard comme la sortie d'erreurs.

Lorsqu'on redirige la sortie standard dans un fichier, les erreurs seront affichées dans la console :

```
cut -d , -f 1 fichier_inexistant.csv > eleves.txt
cut: fichier_inexistant.csv: Aucun fichier ou répertoire de ce type
```

Ici, l'erreur s'est affichée dans la console au lieu d'avoir été envoyée dans eleves.txt.

On pourrait souhaiter enregistrer les erreurs dans un fichier à part : pour cela, on utilise l'opérateur **2>**.

Faisons une seconde redirection à la fin de cette commande cut :

```
cut -d , -f 1 fichier_inexistant.csv > eleves.txt 2> erreurs.log
```

Il y a deux redirections ici :
- **>** redirige le résultat de la commande (sauf les erreurs) dans le fichier eleves.txt (sortie standard)
- **2>** redirige les erreurs éventuelles dans le fichier erreurs.log (sortie d'erreurs)

Il est aussi possible d'utiliser **2>>** pour ajouter les erreurs à la fin du fichier.

### Fusionner les sorties

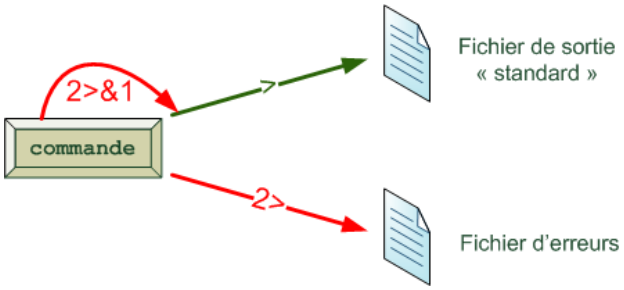
Parfois, on n'a pas envie de séparer les informations dans deux fichiers différents.
Heureusement, il est possible de fusionner les sorties dans un seul et même fichier : avec **2>&1**

Cela a pour effet de rediriger toute la sortie d'erreurs dans la sortie standard.
Traduction pour l'ordinateur : « envoie les erreurs au même endroit que le reste ».

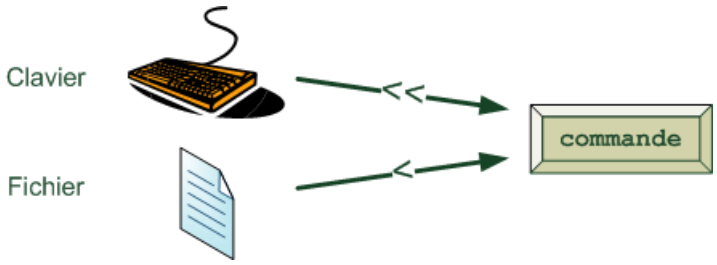
Exemple :

```
cut -d , -f 1 fichier_inexistant.csv > eleves.txt 2>&1
```

Les erreurs seront ajoutées à la fin du fichier eleves.txt comme le reste des messages.



## < et << : lire depuis un fichier ou le clavier




**<** (chevron ouvrant) : permet de lire des données depuis un **fichier** et de les envoyer à une commande

```
cat < notes.csv
```

Écrire cat < notes.csv est strictement identique au fait d'écrire cat notes.csv… du moins en apparence.

Le résultat produit est le même, mais ce qui se passe derrière est très différent.

- `cat notes.csv` : la commande cat reçoit en entrée **le nom du fichier** notes.csv qu'elle doit ensuite se charger d'ouvrir pour afficher son contenu
- `cat < notes.csv` : la commande cat reçoit **le contenu** de notes.csv qu'elle se contente simplement d'afficher dans la console. C'est le **shell** (le programme qui gère la console) qui se charge d'envoyer le contenu de notes.csv à la commande cat.

Ce sont deux façons de faire la même chose mais de manière très différente.

**<<** (double chevron ouvrant) : lire depuis le **clavier** progressivement

Il vous permet d'envoyer un contenu à une commande avec votre clavier.

Exemple :

```
sort -n << FIN
```

La console vous propose alors de taper du texte : on va écrire des nombres, un par ligne (en appuyant sur la touche Entrée à chaque fois); ensuite on va taper **FIN** pour arrêter la saisie.

Tout le texte que vous avez écrit est alors envoyé à la commande (ici sort) qui traite cela en entrée.

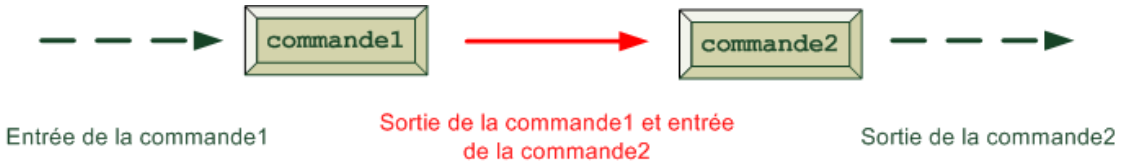
Cela vous évite d'avoir à créer un fichier si vous n'en avez pas besoin.

Il faut définir un **mot-clé** qui servira à indiquer la fin de la saisie (la casse n'est pas importante).

Vous pouvez tout à fait combiner ces symboles avec ceux qu'on a vus précédemment. Par exemple :

```
sort -n << FIN > nombres_tries.txt 2>&1
> 18
> 27
> 1
> FIN
```

## | : chaîner les commandes




Le **pipe** | a le but de chaîner des commandes :
connecter la sortie d'une commande à l'entrée d'une autre commande

En gros, tout ce qui sort de la commande1 est immédiatement envoyé à la commande2.
Et vous pouvez chaîner des commandes comme cela indéfiniment !

Parfois, l'utilité de certaines commandes seules peut paraître limitée, mais celles-ci prennent en général tout leur sens lorsqu'on les combine à d'autres commandes.

Exemples:

- trier les élèves par nom :

```
cut -d , -f 1 notes.csv | sort
```


```
Fabrice,18 / 20,Excellent travail
Mathieu,3 / 20,Nul comme d'hab
Sophie,14 / 20,En nette progression
Mélanie,9 / 20,Allez presque...
Corentin,11 / 20,Pas mal mais...
Albert,20 / 20,Toujours parfait
Benoît,5 / 20,En grave chute
```

```
Fabrice
Mathieu
Sophie
Mélanie
Corentin
Albert
Benoît
```

```
Albert
Benoît
Corentin
Fabrice
Mathieu
Mélanie
Sophie
```

- trier les répertoires par taille :

```
du | sort -nr          # => beaucoup des répertoires
du | sort -nr | head   # => les 10 répertoires les plus gros
du | sort -nr | less   # => naviguer à travers tous les résultats (page par page)
```

- afficher les fichiers qui contiennent le mot "log" dans le dossier /var/log :

```
sudo grep "log" -Ir /var/log | cut -d : -f 1 | sort | uniq
```

  - Grep liste tous les fichiers contenant le mot « log » (`-I` exclut les fichiers binaires)

```
/var/log/installer/syslog:Apr  6 15:14:43 ubuntu NetworkManager: <debug> [1207494883.004888]
/var/log/installer/syslog:Apr  6 15:23:27 ubuntu python: log-output
```

  - Cut extrait de ce résultat uniquement les noms des fichiers
  - Sort tri ces noms de fichiers
  - Uniq supprime les doublons

# Surveiller l'activité du système

## w : qui fait quoi ?

Cette commande me permet de voir d'un seul coup d'oeil si la machine est vraiment surchargée (et si oui, à quel point) et si quelqu'un d'autre est en train d'intervenir sur la machine.

Elle affiche :

- l'heure
- l'up time (aussi accessible via uptime) = depuis combien de temps la machine est en marche
- la charge (aussi accessible via uptime et tload) : la charge moyenne depuis 1min, 5min et 15min
- la liste des connectés (aussi accessible via who) : la liste des personnes connectées sur la machine, ce qu'ils sont en train de faire et depuis combien de temps

## ps, top : lister les processus

**ps** : liste statique des processus lancés par le même utilisateur, dans la même console

ps vous permet d'obtenir la liste des processus qui tournent au moment où vous lancez la commande. Cette liste n'est pas actualisée en temps réel, contrairement à ce que fait **top**.

```
  PID TTY          TIME CMD
23720 pts/0    00:00:01 bash
29941 pts/0    00:00:00 ps
```

PID = numéro d'identification du processus
TTY = le nom de la console depuis laquelle a été lancé le processus
TIME = la durée d'exécution du processus
CMD = le programme qui a généré ce processus

Paramètres:

- **ps** -ef : la liste de tous les processus lancés par tous les utilisateurs sur toutes les consoles
- **ps** -ejH : afficher les processus en arbre (regrouper les processus sous forme d'arborescence)
- **ps** -u <user> : afficher les processus lancés par un utilisateur

**top** : liste dynamique des processus

Top vous permet d'obtenir la liste **interactive** des processus, qui est régulièrement mise à jour.

- il affiche en haut l'uptime et la charge, mais aussi la quantité de processeur et de mémoire utilisée
- il ne peut pas afficher tous les processus à la fois, il ne conserve que les premiers pour qu'ils tiennent sur une « page » de la console
- par défaut, les processus sont triés par taux d'utilisation du processeur

## Ctrl +C, kill : arrêter un processus

**Ctrl + C** = arrêter un processus lancé en console (ne coupe pas le programme brutalement, cela lui demande gentiment de s'arrêter, comme si vous aviez cliqué sur la croix pour fermer une fenêtre)

**kill** <pid> = arrêter (ou tuer) un programme tournant un arrière plan, ayant le numéro d'identification <pid>

```
ps -u mateo21 | grep firefox            # => 32678 ?       00:00:03 firefox-bin
kill 32678
kill -9 32678                           # le paramètre -9 force brutalement le programme à s'arrêter
```

**killall** <cmd> = arrêter (ou tuer) tous les programmes tournant un arrière plan lancés par <cmd>

```
ps -u mateo21 | grep find    # => 675 ... find => 678 ... find => 679 ... find
killall find
```

## halt, reboot : arrêter et redémarrer l'ordinateur

La commande **halt** commande l'arrêt immédiat de l'ordinateur (il faut être root pour arrêter la machine)

```
sudo halt   # => un message sera affiché pour annoncer l'arrêt de l'ordinateur
```

La commande **reboot** pour redémarrer l'ordinateur (il faut être root)

```
sudo reboot   # => le redémarrage prend effet immédiatement
```

Les commandes halt et reboot appellent la commande **shutdown** avec des paramètres spécifiques.

# Exécuter des programmes en arrière-plan

## &, nohup : lancer un processus en arrière-plan

Lorsque vous vous apprêtez à lancer une opération un peu longue (ex: une grosse copie de fichiers), vous n'avez peut-être pas envie de patienter sagement le temps que la commande s'exécute pour pouvoir faire autre chose en attendant.

Certes, on peut ouvrir une autre console.
Mais Il y a des cas où l'on n'a accès qu'à une seule console, ou encore pas envie d'en ouvrir une autre.

Plusieurs programmes peuvent tourner en même temps au sein d'une même console.

**&** : lancer un processus en arrière-plan

La première technique que je veux vous faire découvrir est très simple : rajouter le symbole & à la fin de la commande que vous voulez envoyer en arrière-plan.

```
cp video.avi copie_video.avi &                    # => [1] 16504
find / -name "*log" > sortiefind.txt 2>&1 &       # => [2] 18231
```

Les informations renvoyées :

- [1] = le numéro du processus en arrière-plan dans cette console
- 16504 = le numéro d'identification du processus (le pid)

Attention : les processus ainsi lancés sont "attachés" à la console;
Si vous fermez la console, les processus seront tués et ne s'exécuteront donc pas jusqu'au bout.

**nohup** : détacher le processus de la console

Syntaxe : nohup commande

```
nohup cp video.avi copie_video.avi          # => nohup: ajout à la sortie de `nohup.out'
```

La sortie de la commande est par défaut redirigée vers un fichier nohup.out.
Aucun message ne risque donc d'apparaître dans la console.

La commande continuera de fonctionner quoi qu'il arrive (sauf si on lui envoie un kill).

## Ctrl + Z, jobs, bg & fg : passer un processus en arrière-plan

Tapez **Ctrl + Z** pendant l'exécution d'un programme pour l'arrêter
et pour reprendre immédiatement la main sur l'invite de commandes.

```
cp video.avi copie_video.avi
Ctrl + Z                          # => [1]+ Stopped top
```

Le processus est maintenant dans un état de pause. Il ne s'exécute pas mais reste en mémoire.

Pour passer ce programme en arrière-plan (background), il suffit de taper `bg` (sans paramètre)

```
bg                                # => [1]+ top &
```

Cela commande la reprise du processus, mais cette fois en arrière-plan.
Il continuera à s'exécuter à nouveau, mais en tâche de fond.

Vous pouvez envoyer autant de processus en arrière-plan que vous voulez au sein d'une même console :
- soit en les lançant directement en arrière-plan avec un & à la fin de la commande
- soit en utilisant la technique du Ctrl + Z suivi de bg

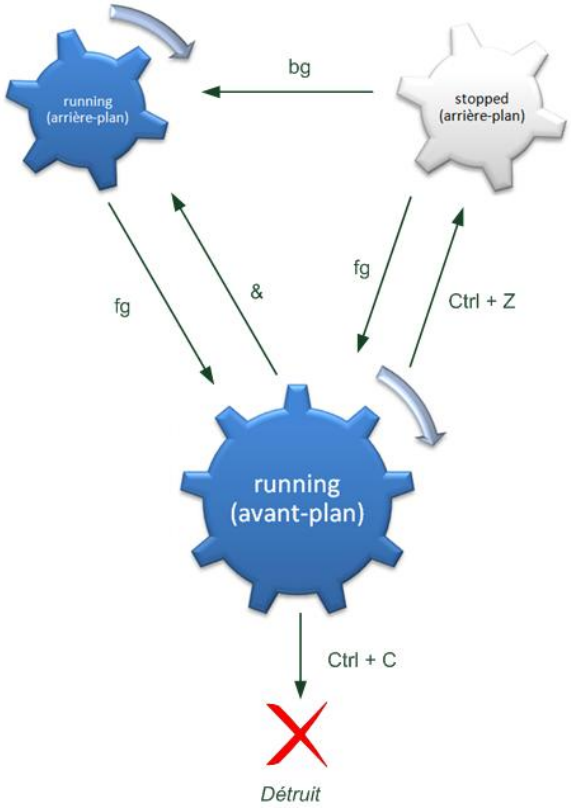
**jobs** : connaître les processus qui tournent en arrière-plan (au sein d'une même console)

```
jobs
[1]-  Stopped   top
[2]+  Stopped   find / -name "*log*" > sortiefind 2>&1
```

**fg** : reprendre un processus au premier plan (foreground)

```
fg                  # reprendre le seul processus lancé en arrière-plan
fg %2               # reprendre le find associé au job numéro 2
```

Si vous avez un seul processus listé dans les jobs, c'est ce processus qui sera remis au premier plan.
Si vous avez plusieurs processus en arrière-plan, il faudra préciser lequel vous voulez récupérer.




Par défaut, un processus est lancé dans l'état **running** à l'avant-plan.

On peut l'arrêter avec la combinaison Ctrl + C, auquel cas il sera **détruit**.

Mais on peut aussi l'envoyer en arrière-plan.

Si on l'exécute dès le départ avec un &, il sera à l'état **running** à l'arrière-plan.

Si on choisit de faire Ctrl + Z, il passera à l'état **Stopped** à l'arrière-plan.

Il faudra taper bg pour le faire passer à nouveau à l'état **running** en arrière-plan.

Enfin, la commande fg renvoie un processus de l'arrière-plan vers l'avant-plan.

## screen : plusieurs consoles en une

**screen** est un multiplicateur de terminal (logiciel libre à installer avec `apt-get install screen`)

Derrière ce nom se cache en fait un programme capable de gérer plusieurs consoles au sein d'une seule, un peu comme si chaque console était une fenêtre !

Paramètres au lancement de screen :

- `screen` : ouvre une nouvelle fenêtre
- `screen -ls` : affiche la liste de tous les screens actuellement ouverts
- `screen -S <sessionname>` : associe un nom à la nouvelle fenêtre qu'on va créer
- `screen -R <sessionname>` : se rattacher à la fenêtre identifié par <sessionname>
- `screen -S <sessionname> -X quit` : tuer la fenêtre identifié par <sessionname>

Sous **screen**, tout se fait à partir de combinaisons de touches de la forme suivante : Ctrl + a touche.
En fait, vous devez taper Ctrl + a, relâcher ces touches et ensuite appuyer sur une autre touche.

Les principales commandes de screen

- Ctrl + a ? : afficher l'aide
- Ctrl + a c : créer une nouvelle fenêtre
- Ctrl + a w : afficher la liste des fenêtres actuellement ouvertes
  En bas de l'écran vous verrez par exemple apparaître : 0-$ bash 1*$ bash.
  Cela signifie que vous avez deux fenêtres ouvertes, l'une numérotée 0, l'autre 1.
  Celle sur laquelle vous vous trouvez actuellement contient une étoile * (ici la fenêtre n° 1).
- Ctrl + a A : renommer la fenêtre actuelle
- Ctrl + a n : passer à la fenêtre suivante (next)
- Ctrl + a p : passer à la fenêtre précédente (previous)
- Ctrl + a Ctrl + a : revenir à la dernière fenêtre utilisée
- Ctrl + a [0-9] : passer à la fenêtre numéro X (entre 0 et 9)
- Ctrl + a ": choisir la fenêtre dans laquelle on veut aller
- Ctrl + a S : découper screen en plusieurs parties (split)
  (pour passer d'une fenêtre à l'autre: Ctrl + A Tab)
  (pour fermer une fenêtre que vous avez splitée : Ctrl + A X)
- Ctrl + a k ou Ctrl + D ou `exit` : **fermer** la fenêtre actuelle (kill)
- **Ctrl + a d** : détache screen, retrouvez l'invite de commandes « normale » sans arrêter screen.
  L'information [detached] apparaît pour signaler que screen tourne toujours et qu'il est détaché de la console actuelle. Il continuera donc à tourner quoi qu'il arrive, même si vous fermez la console dans laquelle vous vous trouvez.
  Vous pouvez revenir récupérer votre session screen plus tard : `screen -r`

Attention : screen est sensible à la casse pour les commandes !

# Exécuter un programme à une heure différée

## date : régler l'heure

```
date                                # => mercredi 10 novembre 2010, 12:27:25 (UTC+0100)
```

Sans paramètre, la commande nous envoie la date actuelle, l'heure et le décalage horaire.

Il est possible de personnaliser l'affichage de la date : quelles informations vous voulez afficher et dans quel ordre (vous pouvez par exemple ajouter les nanosecondes ou encore le numéro du siècle actuel).

Pour spécifier un affichage personnalisé, vous devez utiliser un symbole **+** suivi d'une série de symboles qui indiquent l'information que vous désirez (le tout entre guillemets).

Exemples :

```
date "+%H"                          # => 12 (H = numéro de l'heure actuelle)
date "+%H:%M:%S"                    # => 12:41:01 (H = l'heure; M = les minutes; S = les secondes)
date "+%Hh%Mm%Ss"                   # => 12h41m01s
date "+Bienvenue en %Y"             # => Bienvenue en 2010 ( Y = l'année)
```

Sous root, il est possible de changer la date.
Il faut préciser les informations sous la forme suivante : MMDDhhmmYYYY.
Les lettres signifient : MM : mois ; DD : jour ; hh : heure ; mm : minutes ; YYYY : année (optionnel).

```
sudo date 11101250                  # => mercredi 10 novembre 2010, 12:50:00 (UTC+0100)
```

## at : exécuter une commande plus tard (une seule fois)

Pour exécuter une commande à une heure précise :

- vous indiquez à quel moment (quelle heure, quel jour) vous désirez que la commande soit exécutée.
- vous tapez ensuite la commande que vous voulez voir exécutée à l'heure que vous venez d'indiquer.

Exemple :

```
at 14:17
at> touch fichier.txt               # on vous demande de taper les commandes à exécuter à cette heure-là
at> <EOT>                           # tapez Ctrl + D pour finaliser la demande
                                    # => job 5 at Mon Nov 10 14:17:00 2010
```

Il est possible d'exécuter une commande dans 5 minutes, 2 heures ou 3 jours sans avoir à écrire la date.
Par exemple, pour exécuter la commande dans 5 minutes :

```
at now +5 minutes
```

Les mots-clés utilisables sont les suivants : minutes; hours; days; weeks; months; years.

```
at now +2 weeks
```

Chaque fois qu'une commande est « enregistrée », at nous indique un numéro de job ainsi que l'heure à laquelle il sera exécuté. Il est possible d'avoir la liste des jobs en attente avec la commande **atq** :

```
atq                 # => 13  Mon Nov 10 14:44:00 2010 a mateo21; 12 Mon Nov 10 14:42:00 2010 a mateo21
```

Si vous souhaitez supprimer le job n° 13, utilisez atrm : `atrm 13`

## sleep : faire une pause

Vous pouvez enchaîner plusieurs commandes à la suite en les séparant par des points-virgules :

```
touch fichier.txt; rm fichier.txt
```

touch est d'abord exécuté, puis ce sera le tour de rm (qui supprimera le fichier crée).

Parfois on a besoin de faire une pause entre les deux commandes. C'est là qu'intervient **sleep**.

```
touch fichier.txt; sleep 10; rm fichier.txt
```

Cette fois ci : fichier.txt est créé; sleep fait une pause de 10 secondes; rm supprime ensuite le fichier.

La pause est exprimée par défaut en secondes.
Il est aussi possible d'utiliser d'autres symboles pour changer l'unité : m = minutes; h = heures; d = jours.

```
touch fichier.txt; sleep 1m; rm fichier.txt                    # faire une pause d'une minute
```

Sleep s'assure que la première commande a bien eu le temps de se terminer.

Vous pouvez aussi remplacer les points-virgules par des &&, comme ceci :

```
touch fichier.txt && sleep 10 && rm fichier.txt
```

Dans ce cas, les instructions ne s'enchaîneront que si elles se sont correctement exécutées.
(si touch renvoie une erreur alors les commandes qui suivent ne seront pas exécutées).

## crontab : exécuter une commande régulièrement

Cet outil nous permet de programmer l'exécution régulière d'un programme : toutes les heures, toutes les minutes, tous les jours, tous les trois jours, etc.

**crontab** est en fait une commande qui permet de lire et de modifier un fichier appelé la « crontab ».
Ce fichier contient la liste des programmes que vous souhaitez exécuter régulièrement, et à quelle heure vous souhaitez qu'ils soient exécutés.

**crontab** permet donc de changer la liste des programmes régulièrement exécutés.
C'est toutefois **le programme cron** qui se charge d'exécuter ces programmes aux heures demandées.
Ne confondez donc pas crontab et cron : le premier permet de modifier la liste des programmes à exécuter, le second les exécute.

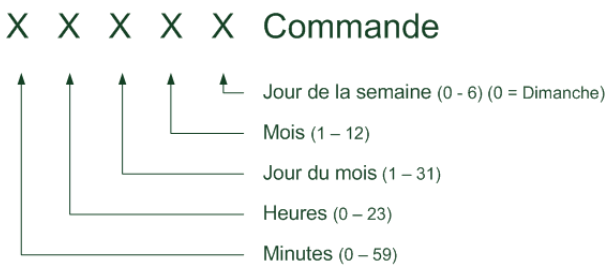
Il y a trois paramètres: **-e** : modifier la crontab; **-l** : afficher la crontab; **-r** : supprimer votre crontab.

```
crontab -l                  # no crontab for mateo21
sudo crontab -l             # no crontab for root
crontab -e                  # ouvre l'éditeur de texte défini par default pour modifier la liste crontab
                            # pour ouvrir nano par default : echo "export EDITOR=nano" >> ~/.bashrc
```

Le fichier ne contient qu'une seule ligne : `# m h  dom mon dow   command`
Cette ligne vous donne quelques indications sur la syntaxe du fichier :

- m : minutes (0 - 59) ;
- h : heures (0 - 23) ;
- dom (day of month) : jour du mois (1 - 31) ;
- mon (month) : mois (1 - 12) ;
- dow (day of week) : jour de la semaine (0 - 6, 0 étant le dimanche) ;
- command : c'est la commande à exécuter.

Chaque ligne du fichier correspond à une commande que l'on veut voir exécutée régulièrement.




Vous devez d'abord indiquer à quel moment vous voulez que la commande soit exécutée, puis ensuite écrire à la fin la commande à exécuter. Chaque « X » sur le schéma peut être remplacé soit par un nombre, soit par une étoile qui signifie « tous les nombres sont valables ».

Exemples:

- exécuter une commande tous les jours à 15h47 :

```
47 15 * * * touch /home/mateo21/fichier.txt
```

  Seules les deux premières valeurs sont précisées : les minutes et les heures.

  Enregistrer et quitter nano : `crontab: installing new crontab`

  Le fichier sera créé dans le répertoire personnel tous les jours à 15 h 47 (s'il n'existe pas déjà).

- `47 * * * * commande` : toutes les heures à 47 minutes exactement (00h47, 01h47, 02h47, etc.)
- `0 0 * * 1 commande` : tous les lundis à minuit (dans la nuit de dimanche à lundi).
- `0 4 1 * * commande` : tous les premiers du mois à 4 h du matin.
- `0 4 * 12 * commande` : tous les jours du mois de décembre à 4 h du matin.
- `0 * 4 12 * commande` : toutes les heures les 4 décembre.
- `* * * * * commande` : toutes les minutes !

La fréquence minimale dans **cron** est toutes les minutes.

Pour chaque champ, on a le droit à **différentes notations** :

- 5 (un nombre) : exécuté lorsque le champ prend la valeur 5 ;
- * : exécuté tout le temps (toutes les valeurs sont bonnes) ;
- 3,5,10 : exécuté lorsque le champ prend la valeur 3, 5 ou 10; ne pas mettre d'espace après la virgule
- 3-7 : exécuté pour les valeurs 3 à 7 ;
- */3 : exécuté tous les multiples de 3 (par exemple à 0h, 3h, 6h, 9h…).

Exemples:

- `30 5 1-15 * * commande` : à 5h30 du matin du 1er au 15eme de chaque mois
- `0 0 * * 1,3,4 commande` : à minuit le lundi, le merdredi et le jeudi
- `0 */2 * * * commande` : toutes les 2 heures (00h00, 02h00, 04h00, ...
- `*/10 * * * 1-5 commande` : toutes les 10 minutes du lundi au vendredi

Si la commande renvoie une information ou une erreur, vous la recevrez par default par e-mail.

```
47 15 * * * touch /home/mateo21/fichier.txt >> /home/mateo21/cron.log 2>&1
```

Tous les messages seront désormais ajoutés à la fin de cron.log

# Archiver et compresser

Format des compressions : zip, rar; gzip, bzip2 (peuvent compresser un seul fichier à la fois); tar.

## tar : assembler des fichiers dans une archive

Deux étapes :

1. réunir les fichiers dans un seul gros fichier appelé **archive** (avec le programme **tar**) ;
2. compresser le gros fichier ainsi obtenu à l'aide de **gzip** ou de **bzip2**.

Les formats **zip** et **rar** ne séparent pas les étapes. Ils sont capables d'assembler plusieurs fichiers en une archive et de la compresser en même temps.

**Créer** une **archive tar** :

```
tar -cvf nom_archive.tar nom_dossier/
```

J'utilise trois options :

**-c** : signifie créer une archive tar ;

**-v** : signifie afficher le détail des opérations ;

**-f** : signifie assembler l'archive dans un fichier.

**Visualiser** le contenu d'une archive tar :

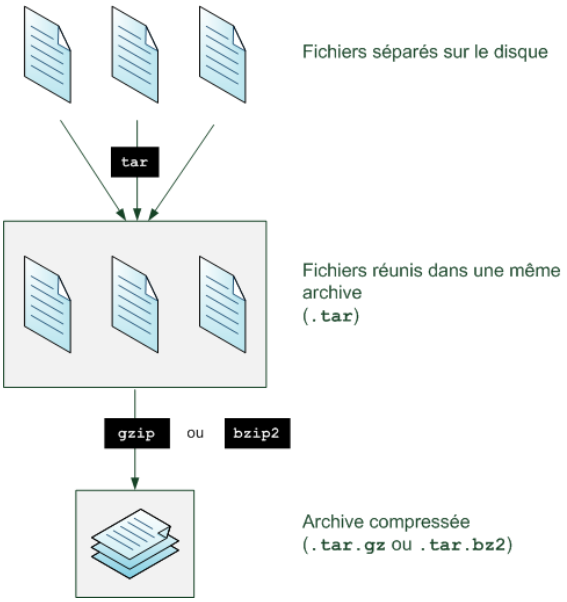
```
tar -tf nom_archive.tar
```

**Ajouter** un fichier à une **archive tar** :

```
tar -rvf nom_archive.tar nom_fichier.txt
```

**Extraire** les fichiers d'une **archive tar** :

```
tar -xvf nom_archive.tar
```




## gzip, bzip2 : compresser une archive

Ces programmes sont simples à utiliser.
Ils prennent comme paramètre le nom du fichier à compresser.
Ils le compressent et modifient ensuite son nom (ils ajoutent un suffixe).

**gzip** : la compression la plus courante; **gunzip** : la décompression des archives gz

```
gzip tutoriels.tar              # => l'archive compressée tutoriels.tar.gz
gunzip tutoriels.tar.gz         # => l'archive non-compressée tutoriels.tar
```

**bzip2** : la compression la plus puissante; **bunzip2** : la décompression des archives bz2

```
bzip2 tutoriels.tar             # => l'archive compressée tutoriels.tar.bz2
bunzip2 tutoriels.tar.bz2       # => l'archive non-compressée tutoriels.tar
```

Il est possible de archiver et de compresser en une seule commande avec **tar**.

Archiver et compresser en gzip avec tar :

```
tar -zcvf tutoriels.tar.gz turotiels/          # archiver et compresser
tar -zxvf tutoriels.tar.gz                     # décompresser
```

Archiver et compresser en bzip2 avec tar :

```
tar -jcvf tutoriels.tar.bz2 turotiels/         # archiver et compresser
tar -jxvf tutoriels.tar.bz2                    # décompresser
```

## unzip, unrar : décompresser les .zip et .rar

unzip : décompresser une archive .zip

```
sudo apt-get install unzip
unzip -l tutoriels.zip                         # visualiser le contenu de l'archive
unzip archive.zip

sudo apt-get install zip
zip -r tutoriels.zip tutoriels/                # créer une archive zip sous linux
```

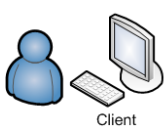
unrar : décompresser une archive .rar (impossible de créer des archives .rar gratuitement sous linux)

```
sudo apt-get install unrar
unrar l tutoriels.rar                          # visualiser le contenu de l'archive
unrar e archive. rar
```

# La connexion sécurisée à distance avec ssh

Toutes les machines sous Linux peuvent être configurées pour que l'on s'y connecte à distance, pour peu qu'elles restent allumées.

## Se connecter à une console à distance





Pour communiquer entre eux en réseau, deux ordinateurs doivent utiliser le même protocole.

I existe de très nombreux protocoles pour que les ordinateurs puissent communiquer entre eux. Il y a : le HTTP (HyperText Transfer Protocol, le protocole utilisé sur le web pour s'échanger des pages web); le FTP (File Transfer Protocol, protocole de transfert de fichiers), l'IMAP (Internet Message Access Protocol, utilisé pour s'échanger des e-mails), le Telnet (un protocole très simple, très basique, qui sert juste à échanger des messages en clair d'une machine à l'autre), le **SSH** (un protocole sécurisé; les données transférées sur le réseau sont chiffrées).

Les méthodes de chiffrement utilisées par le protocole SSH : symétriques et asymétriques.

Le chiffrement **symétrique** :
Avec cette méthode, on utilise une clé (un mot de passe secret) pour chiffrer un message.
Pour déchiffrer ensuite le message, on utilise cette même clé.

Il faut donc que la personne qui chiffre et celle qui déchiffre connaissent toutes deux cette clé qui sert à chiffrer et déchiffrer.

Le chiffrement **asymétrique** :

Utilise une clé dite "publique" pour chiffrer, et une clé dite "privée" pour déchiffrer.

Avec ce type d'algorithme, on ne peut déchiffrer un message que si l'on connaît la clé privée.

L'algorithme de chiffrement asymétrique le plus connu s'appelle **RSA**.

Le chiffrement asymétrique demande beaucoup de ressources au processeur.
Le chiffrement asymétrique est de 100 à 1 000 fois plus lent que le chiffrement symétrique !

SSH utilise les deux chiffrements : asymétrique et symétrique.

Cela fonctionne dans cet ordre.

1. on utilise le chiffrement asymétrique pour s'échanger une clé secrète de chiffrement symétrique
2. ensuite, on utilise tout le temps la clé de chiffrement symétrique pour chiffrer les échanges.

Tout se fait automatiquement. Vous allez juste avoir à entrer un login et un mot de passe pour vous connecter à votre machine à distance.

## Se connecter avec SSH et PuTTy

### Transformer sa machine en serveur

- installer le paquet openssh-server (il va créer automatiquement une clé publique et une clé privée pour les deux algorithmes de chiffrement RSA et DSA)

```
sudo apt-get install openssh-server
```

- le programme de serveur SSH (sshd) sera ensuite lancé automatiquement (et au chaque démarrage)

Lancer et arrêter à tout moment le serveur SSH :

```
sudo /etc/init.d/ssh start
sudo /etc/init.d/ssh stop
```

Le fichier de configuration se trouve dans `/etc/ssh/ssh_config`. Il faudra recharger SSH avec la commande `sudo /etc/init.d/ssh reload` pour que les changements soient pris en compte.

### Se connecter via SSH à partir d'une machine Linux (par default, port 22)

```
ssh <login>@<ip>          # ex : ssh mateo21@87.112.13.165; ssh mateo21@localhost
```

### Se connecter via SSH à partir d'une machine Windows : avec PuTTy (à installer et lancer)

## L'identification automatique par clé

Les deux façons les deux plus utilisées de s'authentifier sur le serveur sont :

- l'authentification par mot de passe ;
- l'authentification **par clés** publique et privée du **client**.

Avec la méthode d'authentification par clés, c'est le client qui va générer une clé publique et une clé privée (=> l'on ne vous demandera pas votre mot de passe à chaque fois pour vous connecter.

### Authentification par clé depuis Linux

Operations sur la machine du **client** :

- générer une paire de clés publique / privée : `ssh-keygen -t rsa`

  => le client génère les clés qu'il va sauvegarder par default sous ~/.ssh/**id_rsa** et ~/.ssh/**id_rsa.pub**
  => on vous demande une passphrase :
  - appuyez sur Entrée pour ne pas chiffrer la clé;
  - ou tapez un mot de passe pour chiffrer la clé sur votre machine (plus sûr)

  Votre dossier SSH contient :
  - **id_rsa** : votre clé privée, qui doit rester secrète (chiffrée si vous avez rentré une passphrase)
  - **id_rsa.pub** : la clé publique que vous devez envoyer au serveur
  - **known_hosts** : c'est la liste des fingerprint que votre PC de client tient à jour. Ça lui permet de se souvenir de l'identité des serveurs et de vous avertir si votre serveur est remplacé par un autre

- envoyer la clé publique au serveur : `ssh-copy-id -i id_rsa.pub <login>@<ip>`

  La clé est ensuite automatiquement ajoutée à ~/.ssh/authorized_keys sur le **serveur**.

Se connecter ensuite au serveur : ssh <login>@<ip> ; entrez votre mot de passe pour la clé si besoin.

Pour eviter de taper votre mot de passe pour la clé à chaque fois : `ssh-add` (il va automatiquement chercher votre clé privée; pour la déchiffrer il vous demandera la passphrase; entrez-la).

## Authentification par clé depuis Windows

Le principe est le même que sous Linux : il faut d'abord que l'on génère une paire de clés sur le PC du client, puis qu'on les envoie au serveur. Nous retrouverons aussi un équivalent de l'agent SSH pour éviter d'avoir à entrer une passphrase à chaque fois.

Pour générer les clés : utiliser Puttygen (instalé avec le programme d'installation de putty)

Copiez la clé publique sur le serveur : `echo "votre_cle" >> authorized_keys`

Les paramètres à configurer en Putty :

- Window → Translation → UTF-8
- Connection → SSH → Auth → Private key file for authentification (Browse)
- Connection → Data → username
- à l'accueil → enregistrer ces paramètres

# Transférer des fichiers

## wget : téléchargement de fichiers directement depuis la console

Il suffit d'indiquer l'adresse HTTP ou FTP d'un fichier à télécharger.

L'option -c demande la reprise d'un téléchargement arrêté.

L'option --background demande un téléchargement en tâche de fond (log dans "wget-log").

```
wget http://.../iso-cd/ debian-40r5-i386-businesscard.iso
wget -c http://.../iso-cd/ debian-40r5-i386-businesscard.iso
wget --background -c http://.../iso-cd/ debian-40r5-i386-businesscard.iso
```

## scp (Secure CoPy) : copier des fichiers sur le réseau

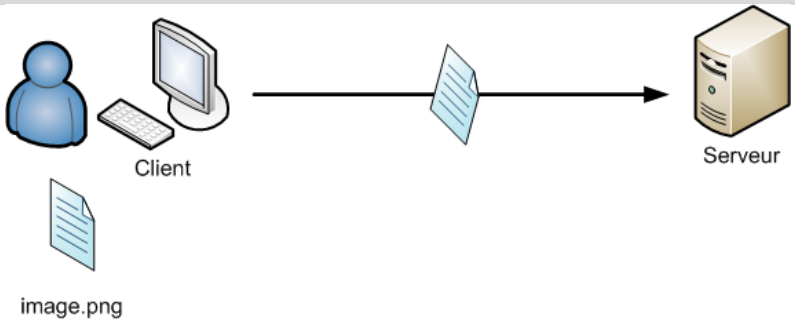
**scp** s'appuie sur ssh pour fonctionner. Là où ssh sert à ouvrir une console à distance (un shell), scp est spécialement conçue pour copier des fichiers d'un ordinateur à un autre.

```
scp <fichier_origine> <copie_destination>
```

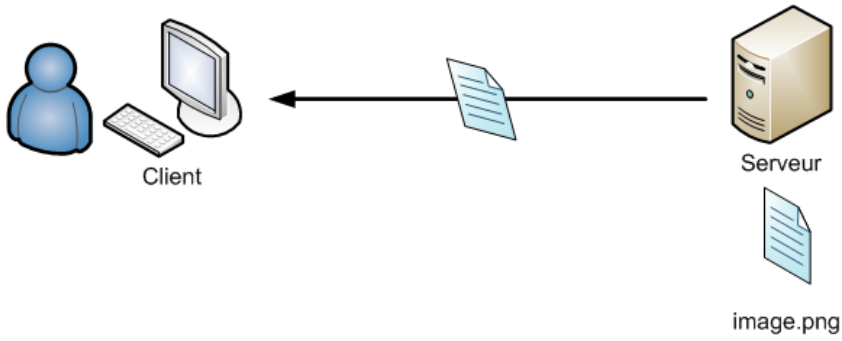
Chacun de ces éléments peut s'écrire sous la forme suivante : **login@ip:nom_fichier**.
Si vous n'écrivez ni login ni IP, scp considérera que le fichier se trouve sur votre ordinateur.
Si le serveur SSH auquel vous essayez de vous connecter n'est pas sur le port standard (22), il faudra indiquer le numéro du port avec l'option **-P**.

Exemples :

```
scp image.png mateo21@85.123.10.201:/home/mateo21/images/
scp image.png mateo21@lisa.simple-it.fr:~/images/
```




```
scp mateo21@85.123.10.201:image.png copie_image_sur_mon_pc.png
scp mateo21@85.123.10.201:image.png .          # copier dans le répertoire dans lequel je me trouve
```



## ftp, sftp : transférer des fichiers

Le FTP (File Transfer Protocol) est un protocole permettant d'échanger des fichiers sur le réseau.

On l'utilise en général dans deux cas :
- pour télécharger un fichier depuis un serveur FTP public (connexion en mode anonyme)
- pour transférer des fichiers vers et depuis un serveur FTP privé (connexion en mode authentifié).

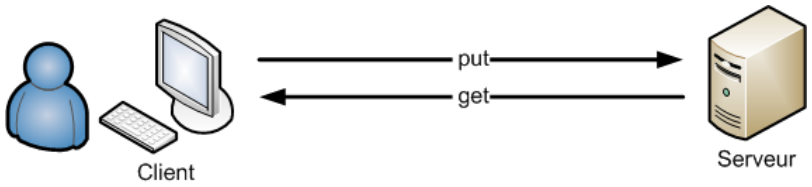
Connexion à un serveur FTP

```
ftp ftp.debian.org          # pour les serveurs ftp publics, le login est "anonymous"
ftp>                        # vous avez ensuite le prompt de ftp> pour rentrer des commandes ftp
```

Se déplacer au sein du serveur ftp : ls, cd, pwd, **!pwd** (le dossier courant sur votre machine), !cd, !ls

Transfert de fichiers : **put** (envoie un fichier vers le serveur), **get** (télécharge un fichier depuis le serveur).

```
ftp> get README             # télécharge README sur votre machine dans le dossier courant
```




**SFTP** : un FTP sécurisé; repose sur SSH pour sécuriser la connexion

```
sftp <login>@<ip>
```

Utilise presque les memes commandes que ftp : put, get, rm, ....

## rsync : synchroniser des fichiers pour une sauvegarde

Il permet d'effectuer une synchronisation (une sauvegarde) entre deux répertoires, que ce soit sur le même PC ou entre deux ordinateurs reliés en réseau.

C'est une sorte de scp intelligent : il compare et analyse les différences entre deux dossiers puis copie uniquement les changements.

Sauvegarder dans un autre dossier du même ordinateur :

```
rsync -arv <nom_dossier_source>/ <nom_dossier_dest>/
```

**-a** : conserve toutes les informations sur les fichiers (droits, date de modification, ...)
**-r** : sauvegarde aussi tous les sous-dossier qui se trouvent dans le dossier à sauvegarder
**-v** : mode verbeux, affiche des informations détaillées sur la copie en cours

Par défaut, rsync ne supprime pas les fichiers dans le répertoire de copie.
Si vous voulez lui demander de le faire, pour que le contenu soit strictement identique :

```
rsync -arv --delete <nom_dossier_source>/ <nom_dossier_dest>/
rsync -arv --delete --backup --backup-dir=/.../backups_supprimes Images/ backups/
```

Sauvegarder sur un autre ordinateur (passer par ssh) :

```
rsync -arv --delete --backup --backup-dir=/.../deleted_files Images/ user@IP:backups/
```

# Analyser le réseau et filtrer le trafic avec un pare-feu

## host, whois : qui êtes-vous ?

Chaque ordinateur relié à l'internet est identifié par une **adresse IP**.

Une adresse IP au format **IPv4** : suite de quatre nombres séparés par des points (ex : 86.172.120.28).
Une adresse IP au format **IPv6** : fe80::209:62fa:fb80:29f2.

On peut associer à chaque IP ce qu'on appelle un nom d'hôte (**hostname**). C'est un nom en toutes lettres plus facile à mémoriser et qui revient exactement au même que d'écrire l'adresse IP.

La commande **host** est capable d'effectuer la conversion dans les deux sens :

- à partir d'une IP on peut avoir le nom d'hôte correspondant ;
- à partir d'un nom d'hôte, on peut avoir l'IP correspondante.

```
host siteduzero.com       # => address 92.243.25.239
host 92.243.25.239        # => domain name pointer lisa.simple-it.fr
```

Etablir une liste de correspondances personnalisée sur votre ordinateur :

```
sudo nano /etc/hosts
127.0.0.1       localhost
127.0.1.1       mateo21-laptop
92.243.25.239   siteduzero.com
192.168.0.5     pc-papa
```

**whois** : tout savoir sur un nom de domaine (qui se trouve derrière : nom, prénom, adresse et moyens de contact; c'est une règle).

```
whois siteduzero.com
```

## ifconfig, netstat : gérer et analyser le trafic réseau

**ifconfig** permet de gérer les connexions réseau de votre machine (ex: pour les activer / désactiver)
**netstat** permet d'analyser ces connexions, de connaître des statistiques, etc.

**ifconfig** : liste des interfaces réseau
Votre ordinateur possède en général plusieurs interfaces réseau (moyens de se connecter au réseau)

```
ifconfig
```

Quelques exemples d'interfaces:

- eth0 : correspond à la connexion par câble réseau (en général le câble RJ45)
- lo : c'est la boucle locale; elle correspond à une connexion à vous-mêmes; tout ce qui est envoyé par là vous revient automatiquement
- wlan0 : il s'agit d'une connexion sans-fil type Wi-Fi

Activer / désactiver une interface : ifconfig <interface> <etat>

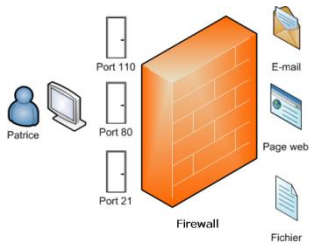
```
ifconfig eth0 down
ifconfig eth0 up
```

**netstat** : statistiques sur le réseau

```
netstat -i      # affiche un tableau présentant une série de statistiques d'utilisation pour chaque interface
netstat -uta    # affiche toutes les connexions ouvertes (-u = UDP; -t=TCP)
netstat -tan    # affiche toutes les connexions TCP ouvertes (-n = affiche les numéros des ports)
netstat -lt     # affiche toutes les connexions TCP à l'état LISTEN (d'écoute)
netstat -s      # affiche des statistiques résumées
```

## iptables : le pare-feu de référence

Il permet d'établir un certain nombre de règles pour dire par quels ports on peut se connecter à votre ordinateur, mais aussi à quels ports vous avez le droit de vous connecter. Le but du pare-feu est d'empêcher que des programmes puissent communiquer sur le réseau sans votre accord.




```
iptables -L             # afficher les règles qui régissent actuellement le pare-feu :
Chain INPUT (policy ACCEPT)     target     prot opt source      destination
Chain FORWARD (policy ACCEPT)   target     prot opt source      destination
Chain OUTPUT (policy ACCEPT)    target     prot opt source      destination
```

Actuellement les règles sont vides : (policy ACCEPT) signifient que, par défaut, tout le trafic est accepté

- INPUT = règles manipulant le trafic entrant
- FORWARD = règles manipulant la redirection du trafic
- OUTPUT = règles manipulant le trafic sortant

Réinitialiser les règles iptables

```
iptables -F             # afficher les règles qui régissent actuellement le pare-feu :
```

Quelques exemples de règles input :

```
iptables -L --line-numbers
Chain INPUT (policy DROP)
num  target     prot opt source               destination
1    ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:www
2    ACCEPT     tcp  --  anywhere             anywhere            tcp dpt:ssh
3     ACCEPT      tcp  --  anywhere               anywhere              tcp dpt:imap2
```

Ajouter et supprimer des règles :

**-A** <chain> : ajoute une règle en fin de liste pour la chain indiquée (INPUT ou OUTPUT)
**-D** <chain> <rulenum> : supprime la règle n° rulenum pour la chain indiquée
**-I** <chain> <rulenum> : insère une règle au milieu de la liste à la position indiquée par rulenum; par default, la règle sera insérée en premier, tout en haut dans la liste.
**-R** <chain> <rulenum> : remplace la règle n° rulenum dans la chain indiquée.
**-F** <chain> : vide toutes les règles de la chain indiquée
**-P** <chain> <regle> : modifie la règle par défaut pour la chain. Cela permet de dire, par exemple, que par défaut tous les ports sont fermés, sauf ceux que l'on a indiqués dans les règles.

# Compiler un programme depuis les sources

## Trouver un paquet .deb

La plupart des programmes dont vous aurez besoin sous Ubuntu sont référencés dans des dépôts et accessibles via la commande apt-get. Toutefois, certains programmes récents ou encore en développement ne sont pas disponibles via apt-get.

Si vous trouvez le **.deb** du programme à installer, téléchargez-le et double-cliquez dessus.

Si aucune erreur n'apparaît vous pouvez procéder à l'installation.

Sinon, cela signifie :

- soit que vous avez téléchargé un .deb ne correspondant pas à votre machine (version 32 bits au lieu de 64 bits ou inversement) ;
- soit qu'il vous manque des dépendances pour pouvoir installer convenablement le programme. Il faut d'abord installer le programme manquant avant d'aller plus loin.

Si même le paquetage .deb n'est pas disponible, il ne reste alors qu'une solution : récupérer le code source du programme et le compiler soi-même. On peut ainsi créer un exécutable spécialement optimisé pour sa machine.

## Quand il n'y a pas d'autre solution : la compilation

La compilation est un procédé qui permet de transformer le code source d'un programme en un exécutable que l'on peut utiliser.

### Compilation d'un programme pas à pas

Pour compiler des programmes, vous aurez besoin avant toute chose d'installer les outils de compilation (dont gcc, ...). Pour cela il suffit d'installer le paquet build-essential :

```
sudo apt-get install build-essential
```

Exemple : installer le programme htop

- télécharger les dernières sources du programme (archve compressée .tar.gz)
- extraire le contenu de l'archive : `tar -zxvf htop-0.8.3.tar.gz`
- se rendre dans le dossier ainsi crée : `cd htop-0.8.3`
- exécuter le programme de **configuration** : `./configure`

  **configure** est un programme qui analyse votre ordinateur et qui vérifie si tous les outils nécessaires à la compilation du logiciel que vous souhaitez installer sont bien présents.

  Son exécution peut prendre du temps car il effectue de nombreux tests.

  Il arrive fréquemment que le programme affiche une **erreur** en raison d'un manque de dépendances.

  **Ex:** `configure: error: missing headers: curses.h.`

  La technique la plus efficace consiste à effectuer une recherche de la ligne d'erreur sur le web.

  L'information à chercher est le nom du paquet manquant que vous devez installer. En lisant les forums, vous devriez finir par trouver le nom du paquet que vous recherchez : **libncurses5-dev**.

  En l'occurrence, il suffit d'installer ce paquet via apt-get : `sudo apt-get install libncurses5-dev`

  Une fois le paquet installé, relancez configure (répéter le processus ci-dessus en cas d'erreur).

- lancer la **compilation**: `make`
- **installer** le programme (le copier dans le bon répeertoire): `sudo make install`
- **lancer** le programme : `htop`

# Vim : l'éditeur de texte du programmeur

## Installer vim

Sur la plupart des distributions Linux, **vim** est en général installé par défaut.

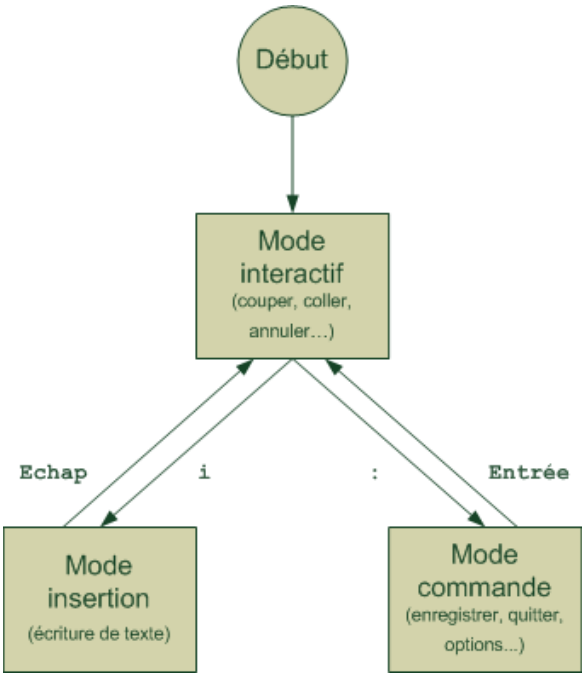
Sinon : `sudo apt-get install vim`

Il existe une vérsion graphique de vim, appellé gVim (même disponible sous Windows).

## Les modes d'édition de vim

Vim possède trois modes de travail différents.

- **Mode interactif** : c'est le **mode par défaut** par lequel vous commencez. Dans ce mode, vous ne pouvez pas écrire de texte. Le mode interactif permet de se déplacer dans le texte, de supprimer une ligne, copier-coller du texte, rejoindre une ligne précise, annuler ses actions, etc. Chaque action peut être déclenchée en appuyant sur une touche du clavier
- **Mode insertion** : permet d'**insérer du texte** à l'endroit où se trouve le curseur. Pour entrer dans ce mode, appuyez sur la touche **`i`** . Pour en sortir, appuyez sur la touche **`<ESC>`**
- **Mode commande** : permet de **lancer des commandes** telles que « quitter », « enregistrer », etc. Vous pouvez aussi l'utiliser pour activer des options de **vim** (comme la coloration syntaxique, l'affichage du numéro des lignes…). Vous pouvez même envoyer des commandes au shell (la console) telles que ls, locate,cp, etc. Pour activer ce mode, vous devez être en mode interactif et appuyer sur la touche deux points « **`:`** ». Vous validerez la commande avec la touche **`<ENTER>`** et reviendrez alors au mode interactif.




NOTE: En appuyant sur **`<ESC>`** on revient dans le mode Interactif ou on annule une commande non désirée et partiellement rempli.

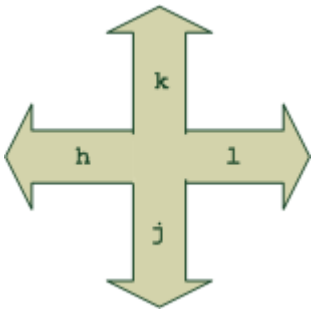
## Lancer vim et executer des opérations basiques et standard

```
vim
```

Pour les nouveaux utilisateurs, **vim** intègre un véritable petit tutoriel : `vimtutor`
**Vimtutor** lance simplement v**im** avec un fichier d'aide prédéfini.

**Raccourcis**:

- lancer vim depuis la console : **vim** <filename> **<ENTER>**
- se déplacer dans le fichier avec les flèches ou avec les touches
  **<h>** (gauche) **<j>** (bas) **<k>** (haut) **<l>** (droite)




- se déplacer en début de ligne : **<0>** ou **<ORIGINE>**
- se déplacer en fin de ligne : **<$>** ou **<FIN>**
- se déplacer de mot en mot : **<w>**
- se déplacer de x mots : **<2w>** , **<5w>** ...
- se déplacer à la fin du fichier : **<ESC> G**
- se déplacer au début du fichier : **<ESC> gg**
- se déplacer à la ligne x du fichier : **<ESC> 7G** , **<ESC> 435G**
- sauvegarder le fichier courant: **<ESC> :w <ENTER>**
- sauvegarder dans un autre fichier: **<ESC> :w** <filename> **<ENTER>**
- quitter vim (aucun changement depuis la dernière sauvegarde) **<ESC> :q <ENTER>**
- quitter vim sans sauvegarder : **<ESC> :q! <ENTER>**
- quitter vim en sauvegardant le fichier : **<ESC> :wq <ENTER>**
- insérer du texte à la position du curseur **<ESC> i** <texte>
- ajouter du texte à la fin de la ligne **<ESC> A** <texte>
- supprimer un caractère à la position du curseur : **<ESC> x** ou **<DEL>**
- supprimer un mot (ou plusieurs) à la position du curseur : **<ESC> dw** , **<ESC> d5w** ...
- supprimer une ligne (ou plusieures) à la position du curseur : **<ESC> dd** , **<ESC> 2dd** ...
- supprimer les caractères jusqu'au début de ligne : **<ESC> d0** ou **<ESC> d<ORIGINE>**
- supprimer les caractères jusqu'à la fin de la ligne : **<ESC> d$** ou **<ESC> d<FIN>**
- remplacer un caractère par un autre : **<ESC> r**<new_char>
- remplacer plusieurs caractères à la suite : **<ESC> R**<new_consecutive_chars>

- changer les caractères jusqu'à la fin du mot (*delete* & *insert*): **<ESC> ce <ESC>**
  - vous pouvez utiliser **cw** pour changer le texte jusqu'au début du mot prochain
  - vous pouvez utiliser **c$** pour changer le texte jusqu'à la fin de la ligne, ...
- copier et coller :
  - **yy** = copier la ligne actuelle en mémoire → **p** = coller sur la ligne situé après le curseur
  - cela fonctionne aussi avec **dd**, qui « coupe » la ligne
  - vous pouvez utiliser **yw** pour copier un mot, **y$** pour copier du curseur jusqu'à la fin de la ligne, ...
- annuler des modifications (*undo* et *redo*):
  - **u** = annuler vos dernières modifications ( les dernières 4 modifications : **4u** )
  - **CTRL+R** = annuler une annulation
  - **U** = annuler tous les changements faits sur une ligne

## Opérations avancées (recherche, split, fusion, ...)

Rechercher un mot (**/**<mot> ou **?**<mot>) :

- tapez **/** ou **?** pour passer en mode recherche : le curseur se place en bas de l'écran
- écrivez ensuite le mot que vous recherchez
- tapez ensuite sur **<ENTER>**

Le curseur se place alors sur la première occurrence du mot recherché dans le fichier (avec **?** la recherche s'effectue du début du fichier; avec **/** la recherche s'effectue à partir de la position courante).

Pour passer à la prochaine occurrence du mot : appuyez sur **n**
Pour passer à la précédente occurrence du mot : appuyez sur **N**

Rechercher et remplacer un mot :

- **:s/**<ancien>**/**<nouveau> : remplace la 1ère occurrence de la ligne où se trouve le curseur
- **:s/**<ancien>**/**<nouveau>**/g** : remplace toutes les occurrences de la ligne où se trouve le curseur
- **:#,#s/**<ancien>**/**<nouveau>**/g**: remplace toutes les occurrences dans les lignes n° # à # du fichier
- **:%s/**<ancien>**/**<nouveau>**/g** : remplace toutes les occurrences dans tout le fichier **(!)**

Fusion des fichiers :

Avec **:r <filename>** vous pouvez insérer un fichier à la position du curseur
(l'autocomplétion avec Tab fonctionne là aussi).

Le découpage d'écran (split) :

```
:sp                 # découper l'écran horizontalement; le fichier d'origine est ouvert une seconde fois
:sp <filename>      # découper l'écran horizontalement; le 2eme fichier est ouvert dans le 2eme écran
:vsp                # découper l'écran verticalement
```

Les principaux raccourcis en écran splitté :

- `Ctrl + w` puis `Ctrl + w` : navigue de viewport en viewport (répétez l'opération plusieurs fois)
- `Ctrl + w` puis `j` : déplace le curseur pour aller au viewport juste en dessous. La même chose fonctionne avec les touches `h`, `k` et `l` qu'on utilise traditionnellement pour se déplacer dans Vim.
- `Ctrl + w` puis `+` : agrandit le viewport actuel.
- `Ctrl + w` puis `-` : réduit le viewport actuel.
- `Ctrl + w` puis `=` : égalise à nouveau la taille des viewports.
- `Ctrl + w` puis `r` : échange la position des viewports. Fonctionne aussi avec « `R` » majuscule pour échanger en sens inverse.
- `Ctrl + w` puis `q` : ferme le viewport actuel.

Lancer une commande externe :

Il est possible d'écrire des commandes traditionnelles du shell directement dans Vim : `:! <commande>`
Cette fonctionnalité est bien pratique pour effectuer quelques actions sans avoir à quitter Vim.

## Options de vim

Activer des options dans un fichier de configuration :

```
cp /etc/vim/vimrc  ~/.vimrc
vim .vimrc
syntax on               # activer la coloration syntaxique (pour C, C++, Python, Java, Ruby, Bash, Perl, etc)
set background=dark     # meilleure couleur pour le fond d'écran sous vim
set number              # afficher les numéros de ligne
set showcmd             # afficher la commande en cours
set ignorecase          # ignorer la casse lors de la recherche
set mouse=a             # activer le support de la souris
```

# Introduction aux scripts shell

## Qu'est-ce qu'un shell ?

Il existe plusieurs environnements console (minilangages de programmation intégré à Linux) : les **shells**

- **sh** : *Bourne Shell;* l'ancêtre de tous les shells.
- **bash** : *Bourne Again Shell*; amélioration du *Bourne Shell*, disponible par défaut sous Linux, Mac OS X
- **ksh** : *Korn Shell*; shell puissant assez présent sur les Unix propriétaires, mais aussi disponible en version libre, compatible avec bash.
- **csh** : *C Shell*; un shell utilisant une syntaxe proche du langage C.
- **tcsh** : *Tenex C Shell*; amélioration du *C Shell*.
- **zsh** : *Z Shell*; shell assez récent reprenant les meilleures idées de bash, ksh et tcsh.

Le **shell** fournit toutes les fonctionnalités de base pour pouvoir lancer des commandes.

Le **.bashrc** est le fichier de configuration du bash que Linux vous fait utiliser par défaut.
Chaque personne peut avoir son .bashrc pour personnaliser son invite de commandes, ses alias, etc.

## Le premier script bash

```
vim essai.sh          # extension .sh par convention pour indiquer que c'est un script shell (pas obligatoire)
```

```
#!/bin/bash           # indiquer quel shell est utilisé pour l'exécution; la syntaxe change d'un shell à l'autre
# Affichage de la liste des fichiers     # un commentaire
pwd
ls
```

L'exécuter :

- sauvegarder votre fichier et fermez votre éditeur : **:wq**
- donnez les droits d'exécution au script : **chmod +x** essai.sh
- exécuter le script : **./**essai.sh

```
# =>
/home/mateo21/scripts
essai.sh
```

Exécuter le script en mode debugging : **bash -x** essai.sh

```
# =>
+ pwd
/home/mateo21/scripts
+ ls
essai.sh
```

Pour lancer un programme depuis n'importe quel répertoire, il faut soit le déplacer (ou copier) dans un des répertoires présents dans la liste des répertoires "speciaux" de PATH (`echo $PATH`) ou ajouter un nouvel répertoire de PATH dans le fichier de configuration .bashrc :

```
export PATH=/home/mateo21/scripts:$PATH
```

# Afficher et manipuler des variables

## Déclarer et afficher une variable

Toute variable possède un nom et une valeur.

Pour la définir : <nom_variable>=<valeur> (attention : ne pas mettre d'espaces autour le symbole égal)

```
vim variables.sh
```

```
#!/bin/bash
message="Bonjour tout le monde"          # variable = {nom, valeur}; pas d'espace atour de symbole =
message='Bonjour c\'est moi'
echo $message                            # afficher le contenu de la variable message à l'écran
echo -e "Le message est :\n$message"     # afficher du texte sur plusieurs lignes (\n)
```

### Les quotes

Il est possible d'utiliser des quotes pour délimiter un paramètre contenant des espaces.
Selon le type de quotes que vous utilisez, la réaction de **bash** ne sera pas la même.

Il existe trois types de quotes :

- les **apostrophes** ' ' (simples quotes)

```
message='Bonjour tout le monde'
echo 'Le message est : $message'          # => Le message est : $message
```

  Avec de simples quotes, la variable n'est pas analysée et le $ est affiché tel quel.

- les **guillemets** " " (doubles quotes)

```
message='Bonjour tout le monde'
echo "Le message est : $message"          # => Le message est : Bonjour tout le monde
```

  … ça fonctionne ! Cette fois, la variable est analysée et son contenu affiché.

  En fait, les doubles quotes demandent à bash d'analyser le contenu du message.
  S'il trouve des symboles spéciaux (comme des variables), il les interprète.

- les **accents graves** ` ` (back quotes), qui s'insèrent avec Alt Gr + 7 sur un clavier AZERTY français.

  Les back quotes demandent à bash d'**exécuter** ce qui se trouve à l'intérieur.

```
message=`pwd`
echo "Vous êtes dans le dossier $message"  #=> Vous êtes dans le dossier /home/mateo21/
```

  La commande pwd a été exécutée et son contenu inséré dans la variable message !
  Nous avons ensuite affiché le contenu de la variable.
  Cela peut paraître un peu tordu, mais c'est réellement utile.

## read : demander une saisie

Vous pouvez demander à l'utilisateur de saisir du texte avec la commande **read**.
Ce texte sera immédiatement stocké dans une variable.

La commande **read** propose plusieurs options intéressantes.

La façon la plus simple de l'utiliser est d'indiquer le nom de la variable.

```
#!/bin/bash

read nom
echo "Bonjour $nom !"                    # => Bonjour Mathieu !

read nom prenom
echo "Bonjour $nom $prenom !"            # => Bonjour Deschamps Mathieu !
```

**read** lit ce que vous tapez mot par mot (en considérant que les mots sont séparés par des espaces). Il assigne chaque mot à une variable différente, d'où le fait que le nom et le prénom ont été correctement et respectivement assignés à $nom et $prenom.

Si vous rentrez plus de mots au clavier que vous n'avez prévu de variables pour en stocker, la dernière variable de la liste récupèrera tous les mots restants.

**Options :**

```
#!/bin/bash

read -p "Entrez votre nom : " nom                    # afficher un message de prompt
echo "Bonjour $nom !"                                # => Bonjour Mathieu !

read -p "Votre login (5 chars max) : " -n 5 nom      # limiter le nombre de caractères
echo -e "\nBonjour $nom !"                           # => Bonjour Mathi !

read -p "Votre login (vous avez 5 sec) : " -t 5 nom  # limiter le temps autorisé pour la saisie
echo -e "\nBonjour $nom !"

read -p "Votre mot de passe : " -s pass              # ne pas afficher le texte saisi
echo -e "\nJe vas dire à tout le monde que votre mot de passe est $pass !"
```

## Effectuer des opérations mathématiques

En bash, les variables sont toutes des chaînes de caractères.

En soi, bash n'est pas capable de manipuler des nombres; donc, il ne peut pas effectuer des opérations. Heureusement, il est possible de passer par des commandes; ici, la commande à connaître est **let**.

```
#!/bin/bash

let "a = 5"
let "b = 2"
let "c = a + b"
echo $c                          # => 7

let "a = 5 * 3"                  # $a = 15
let "a = 4 ** 2"                 # $a = 16 (4 au carré)
let "a = 8 / 2"                  # $a = 4
let "a = 10 / 3"                 # $a = 3  (division entière)
let "a = 10 % 3"                 # $a = 1  (modulo; le reste de la division entière)

let "a = a * 3"
let "a *= 3"
```

Bash ne supporte pas les nombre flottants : utilisez d'autres commandes comme **bc**, **awk**, ...

```
c=$(bc <<< "scale=2; $a/$b")
c=$(echo "$a/$b" | bc -l)
c=$(awk "BEGIN {printf \"%.2f\", $a/$b}")
c=$(echo "$a $b" | awk '{printf "%.2f", $1/$2}')
```

## Les variables d'environnement (variables globales)

Les variables d'environnement sont des variables que l'on peut utiliser dans n'importe quel programme. Vous pouvez afficher toutes celles que vous avez actuellement en mémoire avec la commande : `env`

Quelques exemples :

- **SHELL** : indique quel type de shell est en cours d'utilisation (sh, bash, ksh…) ;
- **PATH** : une liste des répertoires qui contiennent des exécutables que vous souhaitez pouvoir lancer sans indiquer leur répertoire. Si un programme se trouve dans un de ces dossiers, vous pourrez l'invoquer quel que soit le dossier dans lequel vous vous trouvez ;
- **EDITOR** : l'éditeur de texte par défaut qui s'ouvre lorsque cela est nécessaire ;
- **HOME** : la position de votre dossier home ;
- **PWD** : le dossier dans lequel vous vous trouvez ;
- **OLDPWD** : le dossier dans lequel vous vous trouviez auparavant.

```
#!/bin/bash

echo "Votre editeur par default est $EDITOR"          # => Votre éditeur par défaut est nano
```

## Les variables des paramètres (argumets ligne de commande)

Comme toutes les commandes, vos scripts bash peuvent eux aussi accepter des paramètres.
Ainsi, on pourrait appeler notre script comme ceci :

```
./variables.sh <param1> <param2> <param3>
```

Pour récupérer ces paramètres dans notre script, des variables sont automatiquement créées :

- **$#** : contient le nombre de paramètres ;
- **$0** : contient le nom du script exécuté (ici ./variables.sh) ;
- **$1** : contient le premier paramètre ;
- **$2** : contient le second paramètre ;
- …

Exemples :

```
#!/bin/bash

echo "Vous avez lancé $0 avec $# paramètres"
echo "Le paramètre 1 est $1"

./variables.sh param1 param2 param3
# => Vous avez lancé ./variables.sh avec 3 paramètres
# => Le paramètre 1 est param1
```

```
#!/bin/bash

echo "Le paramètre 1 est $1"
shift                                  # permet de « décaler » les paramètres dans les variables $1, $2, etc.
echo "Le paramètre 1 est maintenant $1"

./variables.sh param1 param2 param3
# => Le paramètre 1 est param1
# => Le paramètre 1 est maintenant param2
```

## Les tableaux

Le **bash** gère également les variables « tableaux ».

Ce sont des variables qui contiennent plusieurs cases, comme un tableau.

Pour définir un tableau :

```
tableau=('valeur0' 'valeur1' 'valeur2')
```

Cela crée une variable tableau qui contient trois valeurs (valeur0, valeur1, valeur2).

Pour accéder à une case du tableau (cases numérotées à partir de 0):

```
${tableau[2]}
```

Pour définir manuellement le contenu d'une case :

```
tableau[2]='valeur2'
```

Exemples :

```
#!/bin/bash

tableau=('valeur0' 'valeur1' 'valeur2')
tableau[5]='valeur5'
echo ${tableau[1]}                    # => valeur1

tableau=('valeur0' 'valeur1' 'valeur2')
tableau[5]='valeur5'
echo ${tableau[*]}                    # =>  valeur0 valeur1 valeur2 valeur5
```

## Les conditions

Les branchements conditionnels (que nous abrègerons « **conditions** ») constituent un moyen de dire dans notre script « SI cette variable vaut tant, ALORS fais ceci, SINON fais cela ».

### if : la condition la plus simple

```
#!/bin/bash

nom="Bruno"
if [ $nom = "Bruno" ]                    # les  espaces à l'intérieur des crochets sont  obligatoires (!)
then
        echo "Salut Bruno !"
fi                                       # = if à l'inverse; fin if
```

```
#!/bin/bash
nom1="Bruno"
nom2="Marcel"

if [ $nom1 = $nom2 ]
then
        echo "Salut les $nom1 's"
else                                                    # else
        echo "Salut $nom1 et $nom2 !"
fi
```

```
#!/bin/bash

if [ $1 = "Bruno" ]                                     # script avec un paramètre
then
        echo "Salut Bruno !"
elif [ $1 = "Michel" ]                                  # else if
then
        echo "Bien le bonjour Michel"
elif [ $1 = "Jean" ]
then
        echo "Hé Jean, ça va ?"
else
        echo "J'te connais pas, ouste !"
fi
```

### Les tests

Il est possible d'effectuer trois types de tests différents en bash :

- des tests sur des chaînes de caractères :
  - `$v1 = $v2` → identiques
  - `$v1 != $v2` → différentes
  - `-z $v` → chaîne vide; la variable n'existe pas
  - `-n $v` → chaîne non-vide; la variable existe

- des tests sur des nombres

  `$n1 -eq $n2` (*equal*) → valeurs égales
  `$n1 -ne $n2` (*not equal*) → valeurs différentes
  `$n1 -lt $n2` (*lower than)* → valeur inférieure
  `$n1 -le $n2` (*lower or equal*) → inférieure ou égale
  `$n1 -gt $n2` (*greater than)* → superior
  `$n1 -ge $n2` (*greater or equal)* → supérieure ou égale

- des tests sur des fichiers

  `-e $file` (*exists*) → vérifie si le fichier existe
  `-d $file` (*directory*) → vérifie si le fichier est un répertoire
  `-f $file` (*file*) → vérifie si le fichier est un fichier
  `-L $file` (*link*) → vérifie si le fichier est un lien symbolique
  `-r $file` (*read*) → vérifie si le fichier est lisible
  `-w $file` (*write*) → vérifie si le fichier est modifiable
  `-x $file` (*execute*) → vérifie si le fichier est exécutable
  `$file1 -nt $file2` (*newer than)* → vérifie si le fichier est plus récent
  `$file1 -ot $file2` (*older than)* → vérifie si le fichier est plus vieux

Effectuer plusieurs tests à la fois : et <&&>, ou <||>

```
if [ $# -ge 1 ] && [ $1 = 'koala' ]
then
        echo "Bravo !"
        echo "Vous connaissez le mot de passe"
else
        echo "Vous n'avez pas le bon mot de passe"
fi
```

Inverser un test

Il est possible d'inverser un test en utilisant la négation : avec le point d'exclamation « ! »

```
if [ ! -e fichier ]
then
        echo "Le fichier n'existe pas"
fi
```

## Case : le "switch" du bash

```
case $1 in
       "Bruno")                                  #  "B*" = accepte tous les noms qui commencent par B
                echo "Salut Bruno !"
                ;;                               # ;; demande à bash d'arrêter la lecture du case
      "Michel")
                echo "Bien le bonjour Michel"
                ;;
       "Jean" | "Marc" | "Alex")       # plusieurs options possibles (séparées par < | >)
                echo "Salut fréro !"
                ;;
       *)                                        # le "else" du case : si aucun test n'a été vérifié
                echo "J'te connais pas, ouste !"
                ;;
esac                                             #  fin du case
```

## Les boucles

Ces structures permettent de répéter autant de fois que nécessaire une partie du code.

### while : boucle << tant que >>

```
while [ -z $reponse ] || [ $reponse != 'oui' ]
do
        read -p 'Dites oui : ' reponse
done
```

Tant que l'un des deux tests est vrai, on recommence la boucle.
Equivalent à : « Tant que la réponse est vide ou que la réponse est différente de oui ».

### for : boucler sur une liste de valeurs (equivalent à foreach)

```
for variable in 'valeur1' 'valeur2' 'valeur3'
do
        echo "La variable vaut $variable"
done
```

```
liste_fichiers=`ls`
for fichier in $liste_fichiers                # equivalent à    for fichier in `ls`
do
        echo "Fichier trouvé : $fichier"
        mv $fichier $fichier-old
done
```

Simuler un **for classique** :

```
for i in `seq 1 10`
do
        echo $i
done
```

**TP** : créer un script multirenommage.sh qui va rajouter le suffixe "-old" uniquement aux fichiers qui correspondent au paramètre envoyé par l'utilisateur. Si aucun paramètre n'est envoyé, vous demanderez à l'utilisateur de saisir le nom des fichiers à renommer

```
#!/bin/bash

if [ $# -ge 1 ]
then
       liste_fichiers=`ls $1`
else
       read -p "Donnez le nom des fichiers à renommer" fichiers
       liste_fichiers=`ls $fichiers`
fi

for fichier in $liste_fichiers
do
        echo "Fichier trouvé : $fichier"
        mv $fichier $fichier-old
done
```