UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Licence Info
Info0302
2018/2019

Linux : généralités et commandes de base

L'enseignement d'Info0302, "**Stage Unix - *scripting***", est dispensé en L2 INFO et L3 INFO-Pass. Il a pour objectif d'aborder toutes les bases concernant Linux nécessaires pour suivre un cursus de Licence d'Informatique :

- Généralités et commandes de bases
- *shell-scripts*

Sans prétention d'être exhaustif, ce support traite de la première des deux parties. Les exemples et captures d'écran sont issus d'une **Ubuntu 14.04**, virtualisée ; le shell utilisé est `bash`.

Auteur : Ch Jaillet, département MMI, UFR Sciences, Université de Reims Champagne-Ardenne

---

Bibliographie : Il existe de nombres supports, souvent de bonne qualité. Je me suis inspiré de certains d'entre eux :

1. Webographie principale :
   - "Reprenez le contrôle à l'aide de Linux", M Nebra [Openclassroom]
   - "Cours Linux", S Cherrier [site personnel]
     => `http://sylvain.cherrier.free.fr/documentations/coursLinux.pdf`
   - "Cours Linux - Installation et administration", Atrid (??) [`cours-gratuits.com`]
2. Mementos :
   - commandes :
     "Cours Linux : les commandes Linux", Benjamin (??) [`generation-linux.fr`]
   - scripts :
     "Aide mémoire des commandes Bash",?? [`misfu.com/`]
     => `http://www.misfu.com/commandes-bash.html`
3. Pour ceux qui aiment le papier :
   - assez en phase avec l'objectif de ce support : "Shell sous Unix/Linux : Apprenez à écrire des scripts pour administrer votre système", S Maccagnoni [ENI]
   - plus complet (Linux) : "Linux. Administration système et réseau", M Pybourdin & Co, Laboratoire des technologie Linux [Supinfo]
   - plus complet (systèmes) : "Linux. Programmation système et réseau", J Delacroix [Sciences Sup]

Remerciements : - à P Mignot, qui a rédigé des supports très pointus pour un cours similaire il y a quelques années, et que nous utiliserons notamment concernant le *scripting* Linux

- à S Rampacek et H Baala, qui ont rédigé de nombreux énoncés d'exercices et m'ont donné le droit de les réutiliser librement.

Table des matières

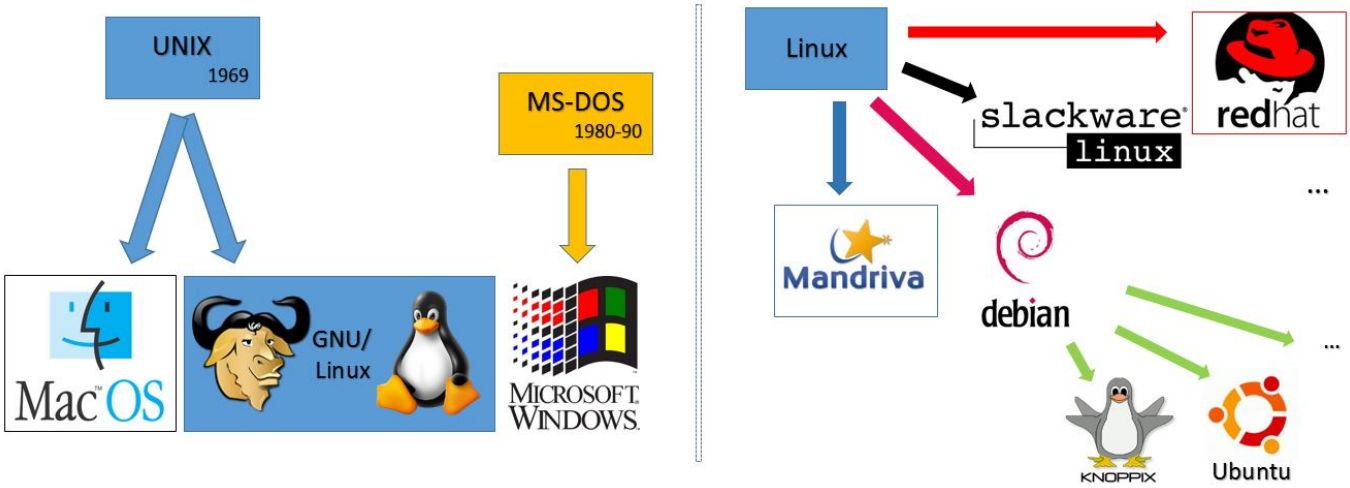
1 Généralités

1.1 Distributions Linux

Linux est un système d'exploitation. Il est gratuit et réputé pour sa sécurité et ses mises à jour fréquentes.

Il s'agit d'une version libre d'**UNIX**, développée à partir de 1984 dans le projet **GNU**. Le noyau a été développé à partir de 1981 par Linus Torvalds.

Il existe différentes distributions, qui diffèrent essentiellement par leur éditeur (entreprise qui mène le projet), leur mode d'installation (simplifiée, personnalisée, ...), la gestion de l'installation des programmes (paquetages RPM, paquetages DEB, archives) et les programmes pré-installés.



La caractéristique de la distribution **Debian** est d'avoir été développée par des développeurs indépendants et non pas par une entreprise. **Ubuntu** est une version grand public de Debian, mise à jour environ deux fois par an, avec un support très important pour la communauté des utilisateurs. Cela en fait le distribution la plus populaire à l'heure actuelle.

En mode graphique, il existe différentes présentations possibles (gestionnaires de bureau Gnome, KDE, Unity, ...), ou en mode console. Par exemple, Ubuntu se décline en Kubuntu (bureau KDE), Xubuntu (XFCE), ... L'environnement par défaut d'Ubuntu, appelé également Ubuntu, est basée sur (Unity)[1].

On peut installer Linux seul sur une machine, en parallèe d'un autre système (il faut alors un lanceur : Grub / ...), ou virtualisé à l'intérieur d'un autre système d'exploitation par l'intermédiaire d'un logiciel de virtualisation (VirtualBox / ...) : dans ce cas Linux est un *système invité*, qui partage les ressources du *système hôte* : "*Mon Ubuntu tourne dans Windows*".

1.2 Caractéristiques de Linux

Les systèmes Unix ont les caractéristiques suivantes, qui les rendent fiables et performants :

1. A partir de Ubuntu 18, la distribution standard est livrée avec un environnement Gnome.

multi-utilisateur : plusieurs utilisateurs peuvent travailler en même temps en se connectant au système (utilisation locale ou distante)

multi-tâche : les programmes s'exécutent en temps partagé sur le(s) processeur(s)

sécurisé : un système de groupes gère les droits des utilisateurs sur les ressources matérielles et logicielles et sur les fichiers : seul l'administrateur peut exécuter des actions d'administration (sensibles) ; les utilisateurs ont chacun un espace personnel et disposent du contrôle de l'accès à leurs fichiers ; les utilisateurs font partie de groupes, dont les droits sont spécifiques (différents groupes peuvent avoir des droits différents).

conçus pour travailler en réseau : ils supportent les protocole d'accès distant (telnet, ssh, ...) et de transfert de fichier (ftp, sftp, ...)

1.2.1 Travail en mode console

On peut utiliser Linux en mode graphique ou en mode console.

- Lorsque vous vous connectez à un serveur Unix, vous travaillez par l'intermédiaire d'un terminal et *a priori* à distance : vous pouvez lancer des applications graphiques sur le serveur, dont les fenêtres s'afficheront si la connexion est réalisée par un protocole qui permet de faire transiter le vidéo (*export* du *display*), mais vous travaillerez certainement plutôt en mode console.
- Lorsque vous vous connectez sur un système Linux, localement, vous arrivez dans le système en mode graphique. Mais vous pouvez également travailler en mode console en ouvrant un **Terminal** (ou plusieurs), ou en basculant en mode console : en effet vous disposez de 6 terminaux, indépendants de l'environnement fenêtré du système. Il existe des raccourcis pour basculer entre ces 7 environnements :
  1. `Ctrl+Alt+F1` : terminal 1 (*tty1*)
  2. `Ctrl+Alt+F2` : terminal 2 (*tty2*)

     ...
  6. `Ctrl+Alt+F6` : terminal 6 (*tty6*)
  7. `Ctrl+Alt+F7` : retour au système fenêtré

  Vous pouvez basculer entre ces environnements et y lancer des programmes de façon concurrente, qui peuvent même communiquer (!!).

1.3 L'arborescence

Dans les systèmes Unix, il n'existe pas de système de disques portant des noms de lecteurs indépendants. Tout est rassemblé en une unique arborescence, où les disques durs physiques sont *montés* comme s'il s'agissait de répertoires (de même pour les lecteurs de CD/DVD, les clés USB, ...). La **racine**, notée `/` est la base de cette arborescence.

Comme dans les autres organisations de fichiers et dossiers/répertoires, on dispose de chemins relatifs et absolus pour se repérer dans l'arborescence, mais avec deux différences :

- la notation absolue part nécessairement de **la** racine
- pour séparer le contenant du contenu on utilise le symbole `/` (slash) (pas l'anti-slash \ utilisé sous Windows) : ainsi si on se trouve dans un répertoire `rep0`, qui contient un répertoire `rep1`, lui-même contenant le fichier `fic`, on désigne `fic` par `rep1/fic` ou `./rep1/fic`, voire `../rep0/rep1/fic` (de même que sous Windows, `.` désinge le répertoire courant et `..` le répertoire parent).

1.3.1 L'organisation du système

Classiquement, les systèmes Unix ont la même arborescence [à peu de choses près]. En considérant les principaux répertoires à la racine, on peut décrire l'organisation les principaux éléments du système (rassemblés ici selon leur fonction) :

`/boot` : contient les fichiers invoqués au démarrage de Linux

`/etc` : contient des fichiers de configuration du système

`/proc` : contient des informations matérielles sur la machine / le serveur (ensemble de fichiers décrivant le système : nombre de processeurs/cœurs et leurs caractéristiques, volume et caractéristiques de la mémoire, ...)

`/dev` : c'est là que se trouvent les dossiers qui représentent les périphériques (lecteurs de CD/DVD, ...), les fichiers représentant les terminaux (*tty1*, *tty2*, ...[2]), ...
Notez qu'on y trouve également `stdin`, `stdout` et `stderr`, descripteurs de fichiers pour l'entrée standard, la sortie standard et la sortie d'erreur standard (par défaut la clavier, l'écran et l'écran à nouveau) que nous évoquerons à nouveau pour réaliser des *redirections*.

`/media` : on peut y monter les dossiers correspondant aux périphériques amovibles (clé USB, carte mémoire, ...) (le système peut être paramétré pour que leur montage soit automatique lorsqu'on les branche)

`/mnt` : on peut également y monter des périphériques, *a priori* de façon plus temporaire

`/bin` : contient les programmes exécutables disponibles pour l'ensemble des utilisateurs

`/sbin` : contient les programmes exécutables disponibles pour **root** (super-utilisateur du système)

`/usr` : c'est un répertoire contenant les programmes installés par les utilisateurs (s'ils ont le droit de les installer ou si root le fait pour eux)

`/lib` : (pour *library*) dossier contenant les bibliothèques partagées (généralement des fichiers `.so`) utilisées par les programmes, y compris ceux que vous écrirez (l'équivalent des `.dll` utilisés sous Windows)

`/opt` : répertoire utilisé pour les add-ons de programmes (*plugins* en anglais, qui permettent d'ajouter des fonctionnalités aux programmes existants)

`/home` : (ou `/usr/home`) contient les répertoires personnels des utilisateurs (leur espace de travail, où ils ont tous les droits sur leurs fichiers) : vous y avez vos répertoires de connexion (portant votre nom d'utilisateur, qui sera également noté `~` pour vous lorsque vous serez connecté[3]

`/root` : c'est le dossier personnel de l'utilisateur « root » (notez qu'ils n'est pas dans `/home` ou `/usr/home`)

`/tmp` : dossier utilisé pour stocker des fichiers [temporaires] des programmes

`/var` : ce dossier contient des données *variables*, notamment les *logs* enregistrés pour tracer l'utilisation de la machine

2. observez leur nombre...

3. On peut donc considérer que `/`, mais aussi que `~`, peuvent tous deux être utilisés pour désigner des chemins absolus de fichiers ou répertoires.

1.3.2 Les différents types de fichiers Unix

Dans les systèmes Unix tout est fichier : les fichiers dits *réguliers* sont ceux qui stockent de l'information, mais les répertoires sont également considérés comme des fichiers, et il existe précisément six types de fichiers Unix, chacun repéré par un symbole :

	Signification	**Rôle**	**Exemples**
-	Fichier régulier	stockage de données	archives (`.zip`, `.tar.gz`, `.deb`, ...) fichiers multimédias (`.avi`, `.mp3`, ...) documents (`.txt`, `.doc`, `.odt`, ...)
d	répertoire (*directory*)	rassemble des fichiers	`/`, `/home`, `/home/chj`, , ...
l	lien symbolique	"pointe" sur un autre fichier	`/lib/libxtables.so.10`, `/dev/cdrom`
s	*socket*	communiquer par le réseau	`/var/run/proftpd.sock`
b	*block device*	gestion de périphérique de stockage	SCSI/USB/SATA : `/dev/sd*` ou `/dev/sr*` IDE : `/dev/hd*`
c	*character device*	gestion d'autre périphérique (souris, webcam, ...)	souris : `/dev/psaux` carte son : `/dev/dsp` wbecam, carte TV, ... : `/dev/video*`

C'est la commande `file` qui permet de récupérer le type d'un fichier.

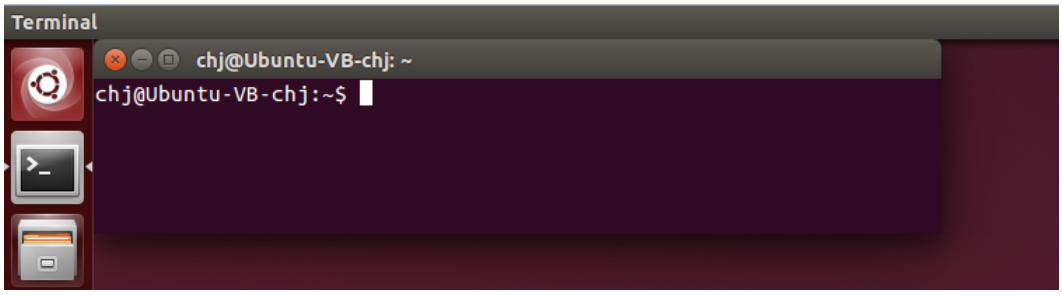
```
$ file /dev/cdrom
```

2 Les fondamentaux

2.1 Avant de commencer

2.1.1 Invite de commande

Lorsque vous ouvrez une console, que ce soit *via* l'application **Terminal** du bureau Unity ou en basculant dans un des terminaux alternatifs, on vous présente une "invite de commande" (sur les terminaux du mode console, il faut vous connecter préalablement).




```
Ubuntu 14.04.5 LTS Ubuntu-VB-chj tty4

Ubuntu-VB-chj login: chj
Password:

Login incorrect
Ubuntu-VB-chj login: chj
Password:
Last login: Tue Aug 28 19:14:19 CEST 2018 on tty4
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.16.0-77-generic i686)

 * Documentation:  https://help.ubuntu.com/

0 packages can be updated.
0 updates are security updates.


WARNING: Security updates for your current Hardware Enablement Stack
ended on 2016-08-04:
 * http://wiki.ubuntu.com/1404_HWE_EOL

There is a graphics stack installed on this system. An upgrade to a
configuration supported for the full lifetime of the LTS will become
available on 2016-07-21 and can be installed by running 'update-manager'
in the Dash.

chj@Ubuntu-VB-chj:~$ cd enseignement/Info0302/
chj@Ubuntu-VB-chj:~/enseignement/Info0302$
```

```
[chj@Ub:~]$ cd enseignement/Info0302
[chj@Ub:Info0302]$ _
```

L'invite de commande (ou *prompt*) est entièrement paramétrable (voir 2.5). Couramment, elle donne le nom de l'utilisateur connecté, le nom de la machine, et le chemin du répertoire courant. Dans le troisième cas j'ai fait en sorte de modifier le texte affiché, par exemple en évitant le nom de la machine et en n'affichant que le nom court du répertoire courant.

2.1.2 Exécuter des commandes

Il suffit de taper la commande dans la console et de valider. Voici quelques exemples, suivant qu'on spécifie ou non des *paramètres* et des *options* (dans chaque cas essayez de deviner l'effet).

1. Commande sans paramètre :

```
[chj@Ub:Info0302]$ cal
```

2. Commande avec paramètre(s) :

```
[chj@Ub:Info0302]$ cal 10
```

```
[chj@Ub:Info0302]$ cal 10 2045
```

3. Commande avec option(s) :

```
[chj@Ub:Info0302]$ cal -h
```

```
[chj@Ub:Info0302]$ cal -h -j
```

```
[chj@Ub:Info0302]$ cal -hj -3
```

4. Commande avec option et paramètre :

```
[chj@Ub:Info0302]$ cal -3 10 2045
```

5. Avec valeur de l'option :

```
[chj@Ub:Info0302]$ cal -A 3
```

```
[chj@Ub:Info0302]$ cal -m 11 -A 3 -B 5
```

6. Avec option(s) ou paramètre(s) long(s) [et éventiellement valeur(s)]

```
[chj@Ub:Info0302]$ date
```

```
[chj@Ub:Info0302]$ date --iso-8601
```

```
[chj@Ub:Info0302]$ date --rfc-2822
```

```
[chj@Ub:Info0302]$ date --date='tomorrow'
```

```
[chj@Ub:Info0302]$ date --date='next monday'
```

```
[chj@Ub:Info0302]$ date --help
```

7. ...

2.1.3 Taper des commandes efficacement

Voici quelques trucs et astuces bien pratiques :

auto-complétion Tapez le début de la commande et utilisez la touche tabulation : s'il n'y a qu'une possibilité l'interpréteur complète ; s'il ne peut pas il vous présente les possibilités. On peut faire de même pour les paramètres des commandes.
Par exemple, dans mon cas, j'ai tapé la commande suivante avec 'c', 'd', ' ', 'e', *tab*, *tab*, 'c', *tab* : en effet il y a trop de commandes commençant par "cd" ; j'ai plusieurs répertoires dans le répertoire courant mais un seul commençant par 'e' ; puis il n'y a qu'un répertoire dans `enseignement` ; *etc*

```
[chj@Ub:~]$ cd enseignement/Info0302/codes/
```

NB : lorsque la liste des possibilités est trop importante, vous pouvez quand même les afficher : la navigation entre les possibilités se fait avec les touches `espace`, `entrée` et `q` (pour quitter) => essayez...

historique de commande Toutes les commandes tapées sont stockées [dans un fichier de votre espace de travail]. Vous pouvez naviguer dans les anciennes commandes avec les flèches du clavier ; et si vous voulez en afficher la liste vous avez justement la commande `history`.

modifier une commande Vous pouvez vous déplacer dans la ligne d'une commande pour la modifier.
Vous pouvez vous aider d'un certain nombre de raccourcis clavier, que je vous laisse tester :

- déplacements : `Ctrl+<`, `Ctrl+>`, `Ctrl+a`, `Ctrl+e`
- couper-coller : `Ctrl+k` ou `Ctrl+u` puis `Ctrl+y` (enlever la fin / le début de la ligne, puis le coller)
- Essayez aussi `Ctrl+l` et même `Ctrl+d`...

rechercher `Ctrl+r` permet de chercher, dans l'historique, les commandes contenant la chaîne de caractères que vous tapez.

2.1.4 Aide sur les commandes

Une connaissance sommaire des commandes est rarement suffisante. C'est par la pratique que vous atteindrez un bon niveau de compétences. En particulier lire **le manuel** est indispensable pour

commencer.
En tout cas si vous vous posez des questions techniques, commencez par lire la documentation. [4]

Obtenir une description sommaire

La commande `whatis` permet d'obtenir un description très sommaire sur la plupart des commandes : en fait plus explicitement elle donne la ligne `NAME` de la page du manuel correspondante (voir plus loin).

La plupart des commandes ont une option `-h`, et/ou `--help`, qui donne quelques informations rapides (mais il faut vérifier...).

```
[chj@Ub:Info0302]$ whatis cal
        ...
[chj@Ub:Info0302]$ cal --help
        ...
[chj@Ub:Info0302]$ cal -h
        ...
[chj@Ub:Info0302]$ whatis calendar
        ...
[chj@Ub:Info0302]$ calendar --help
        ...
[chj@Ub:Info0302]$ calendar -h
        ...
```

La commande `apropos` réalise une recherche inversée : elle parcourt la base de donnée des pages de manuel pour trouver celles contenant l'expression ou les expressions données :

```
[chj@Ub:Info0302]$ apropos cal
        ...
[chj@Ub:Info0302]$ apropos calendar
        ...
[chj@Ub:Info0302]$ apropos de
        ...
[chj@Ub:Info0302]$ apropos installation package
        ...
```

LE manuel

On obtient la documentation sur une commande en utilisant la commande `man` (abréviation de **man**uel).

```
[chj@Ub:Info0302]$ man man
        ...
```

truc : La taille du manuel à afficher dépasse le plus souvent la taille de votre fenêtre de terminal donc `man` n'affiche que le début de la documentation : pour naviguer dans le manuel affiché,

4. On trouve souvent l'expression "familière" *RTFM* : si vous n'en connaissez pas la signification, faites une petite recherche sur le Web par exemple...

utilisez les flèches du clavier, mais également les touches `entrée` et `espace` (avancer d'une linge / d'une page), et `q` pour quitter.
Pour effectuer une recherhe dans le manuel de la commande, utilisez la touche `/` (*slash*) et saisissez le motif à rechercher ; pour passer à l'occurrence suivante, appuyez à nouveau sur la touche `/` (sans resaisir le motif).

Les principales sections présentées dans le manuel d'une commande sont les suivantes (liste non exhaustive) :

- `NAME` : la description très sommaire de la commande (une ligne)
- `SYNOPSIS` : cette section précise la façon d'utiliser la commande : il y a moins de détails que dans la partie `DESCRIPTION` mais cette partie est essentielle pour l'utilisation technique de la commande
  C'est assez compliqué à lire au début mais il faut absolument que vous fassiez l'effort de vous familiariser acec la syntaxe utilisée.
- `DESCRIPTION` : donne la description générale de la commande et la liste détaillée des options, chacune étant particulièrement détaillée
- `AUTHOR`, `COPYRIGHT`, `REPORTING BUGS` : comme leur nom l'indique
- `EXAMPLES` : située le plus souvent à la fin, cette section est bien pratique pour repérer rapidement quelques cas d'utilisation intéressants
- `SEE ALSO` : renvoie à d'autres commandes qui pourraient être intéresssantes pour compléter votre recherche

à noter : nativement tout est en anglais : dans certains systèmes francisés (Ubuntu Fr, ...) les informations peuvent éventuellement être traduites partiellement, mais de toute façon il faut que vous maîtrisiez l'anglais (au moins l'anglais technique, notamment pour lire et comprende les pages de manuel (!!)

truc : si nécessaire, éventuellement au début, vous pouvez quand même installer sur votre système une version française des pages de manuel, avec la commande `apt-get install manpages-fr` (mais méfiez vous de la qualité des traductions)

Il existe des numéros de section qui indiquent au manuel où chercher : en particulier pour résoudre des problèmes de synonymes, vous pouvez préciser la section voulue :

- 1 : les commandes utilisateur
- 2 : les appels système
- 3 : les bibliothèques de programmation
- 4 : les fichiers spéciaux
- 5 : les formats de fichiers
- 6 : les jeux
- 7 : divers
- 8 : les commandes d'administration
- 9 : le noyau

```
[chj@Ub:Info0302]$ man printf
        ...
[chj@Ub:Info0302]$ man 3 printf
        ...
```

```
[chj@Ub:Info0302]$ man kill
        ...
[chj@Ub:Info0302]$ man 2 kill
        ...
```

Autres sources de documentation

Il existe de nombreuses ressources, notamment sur le Web. La communauté Linux est très active et gère en particulier de nombreux forums de discussion et d'entraide. Voici donc des moyens alternatifs pour obtenir de l'information / vous tenir au courant :

Texinfo : On peut utiliser la commande `info` à la place de la commande `man`

docs et **HowTo** : En principe le répertoire `/usr/doc` contient de la documentation sur les principaux packages installés. En particuliers dans `/usr/doc/HOWTO` vous pourrez trouver des explications sur la configuration de votre système. On en trouve une version francisée en ligne à l'adresse `http://www.freenix.fr/linux/HOWTO/`

communauté(s) :

The Linux Documentation Project (LDP) : `http://tldp.org/`
Unix Guru Universe (UGU) : `http://www.ugu.com/`
Linux Center : `http://www.linux-center.org/fr/`
Documentation Ubuntu-fr : `https://doc.ubuntu-fr.org/`

2.2 Les commandes de base

Les premières commandes utiles pour travailler dans un système Unix en mode console concernent le gestion de votre arborescence de travail. On va donc évoquer ici les principales commandes de base, qui concernent les répertoires et les fichiers.

2.2.1 Règles de nommage des fichiers et répertoires

- par défaut les fichiers/répertoires dont le nom commence par un point ne sont pas affichés : pour "cacher" un fichier, faites commencer son nom par un point
- tous les caractères et tous les noms de fichiers sont possibles [à part les caractères de contrôle utilisés dans les expressions régulières (voir ci-dessous)]
  Eviter les noms de fichiers commençant par `-`, et comme nom de fichier exécutable des noms de commande Unix ou shell (par exemple `test`)

2.2.2 Expressions régulières pour les noms de fichiers et répertoires

Certains caractères, dits *spéciaux*, peuvent être utilisés dans des *expressions régulières* : ils permettent de former une chaîne de caractères désignant un ensemble de fichiers ou répertoires (nom générique).

Lorsqu'une commande contient une expression régulière, le shell interprète l'expression régulière, la remplace dans la commande par une liste de fichiers ou de répertoire, puis il exécute la nouvelle commande ainsi construite.

Requêtes sur des fichiers existants

Pour les expressions régulières suivantes, l'expression régulière n'est étendue que sur les fichiers existants correspondants à l'expression régulière.

Définition

`*`	toute chaîne de caractères
`?`	tout caractère
`[aeyuio]`	toute lettre parmi la liste {a ; e ; y ; u ; i ; o}
`[!aeyuio]`	tout caractère autre que {a ; e ; y ; u ; i ; o}
`[a-f]`	les lettres de a à f

Exemples

`ab*`	tout fichier commençant par ab
`*ab`	tout fichier finissant par ab
`a*b`	tout fichier commençant par a et finissant par b
`a??`	tout fichier de 3 caractères commençant par a
`[a-z]*.[cho]`	tout fichier commençant par une lettre minuscule et dont l'extension est `.c`, `.h` ou `.o`

Construire une requête plus générale

{$expr_1, \ldots, expr_n$}	$expr_1$ ou $expr_2$ ou ... ou $expr_n$

Exemples

`a{1,2}`	les fichiers `a1` et `a2`
`a{1,b{A,B}}`	les fichiers `a1`, `abA`, `abB`

Différence entre ces deux écritures

`ls -l a[1-3]`	liste les fichiers existants parmi `a1`, `a2`, `a3`
`ls -l a{1,2,3}`	liste les fichiers `a1`, `a2` et `a3` une erreur est signalée si l'un de ces fichiers n'existe pas
`mkdir a[1-3]`	création du répertoire `a[1-3]`
`mkdir a{1,2,3}`	création des répertoires `a1`, `a2` et `a3`

2.2.3 Manipulation des fichiers et répertoires

L'arborescence (répertoires)

Les répertoires ont ceci de particulier qu'on peut s'y déplacer (les *traverser*). Les premières commandes concernent exclusivement les fichiers.

`cd` (*change directory*) : permet de se déplacer dans l'arborescence

- `cd chemin` pour accéder au répertoire dont on donne le chemin (chemin absolu ou relatif)

- en particulier `cd ..` permet d'accéder au répertoire racine et `cd /` d'accéder à la racine du système
- sans paramètre, cette commande permet à l'utilisateur de revenir à son répertoire personnel : `cd` et équivalent à `cd ~`

`ls` (*list* [*the content of a directory*]) : affiche le contenu du répertoire courant ou du répertoire dont le nom est passé en paramètre : cette commande en liste les **entrées** (fichiers et répertoires)
Cette commande peut prendre un ou plusieurs paramètres :

- on l'utilise sans paramètre pour lister le contenu du répertoire courant
- on peut afficher le contenu de plusieurs répertoires (à la suite)
- avec un/des masque(s) de noms de fichiers on peut afficher la liste des fichiers correspondant à certains critères ; il est également possible d'utiliser des masques pour les noms de répertoires

Cette commande possède de nombreuses options, dont les principales sont

`-a` (*all*) affiche toutes les entrées, y compris les fichiers cachés et les répertoires cachés. En particulier, on y voit `.` et `..` , répertoire courant et répertoire parent

`-l` (*long*) pour l'affichage détaillé : avec des détails sur le type de fichier, les droits (voir 2.3), la taille, ... Déterminez quels sont les différents champs présentés.

Testez les différentes options, en particulier la combinaision `-al`, *etc.*

```
[chj@Ub:~]$ ls -al Doc*/*.txt enseign*/Info0???/projet/{[mM]akefile,*.[hco]}
```

`pwd` (*print working directory*) : affiche le chemin complet du répertoire courant

`mkdir` (*make a directory*) : crée un/des répertoire(s)

`rmdir` (*remove a directory*) : supprime un/des répertoire(s)
Le(s) répertoire(s) doit/doivent être vide(s).

```
[chj@Ub:~]$ mkdir enseignement/Info0301
        ...
[chj@Ub:~]$ cd enseignement/Info0301
        ...
[chj@Ub:Info0301]$ mkdir TP1 TP1/exo1 TP2 TP13
        ...
[chj@Ub:Info0301]$ ls -a TP? ../Info0302/*
        ...
[chj@Ub:Info0301]$ cd
        ...
[chj@Ub:~]$ rmdir enseignement/Info0301/TP13 ennseignement/Info0301
        ...
```

`rm`, `mv`, **...** Bien sûr il existe d'autres commandes qui concernent les répertoires, mais sont plus générales puisqu'elles concernent également les fichiers réguliers : nous les aborderons immédiatement, dans la section suivante.

anecdotique ? Si vous devez réaliser des traitements sur un ensemble de répertoires, il est possible d'empiler les noms des répertoires, puis de les dépiler au fur et à mesure qu'on les a traités : les commandes sont `pushd`, `dirs` et `popd` [5]

- `pushd` : empiler le/les répertoire(s) mentionné(s) en paramètre
- `dirs` : afficher la lite des noms de répertoires empilés
- `popd` : dépiler le nom du répertoire (pas de paramètre : l'action ne dépend que de l'état de la pile)

NB1 : lorsqu'on utilise la commande `pushd`, on se déplace dans le répertoire mentionné ; de même lorsqu'on dépile on se déplace ensuite dans le répertoire au sommet de la pile

NB2 : lorsqu'on utilise la commande `cd` on ne détruit pas la pile, mais on change son sommet au fur et à mesure des déplacements [6]

à noter : bien que cela concerne plus la gestion des disques que des répertoires, mentionnons tout de même les deux commandes `df` et `du` qui permettent de vérifier la taille de l'espace de stochage :

- `df` (*disk free*) : espace disponible sur les différents espaces de fichier de la machine
- `du` (*disk usage*) : espace utilisé dans les différents répertoires du répertoire courant (ou du répertoire dont le nom est passé en paramètre)

Les fichiers (répertoires et fichiers réguliers)

La commande `touch` permet de créer un ou plusieurs fichier(s) régulier(s). Si on l'utilise avec un nom de fichier existant, elle met à jour la date de dernière modification du fichier (à la date du moment).
Vous pourrez vérifier qu'elle permet de mettre à jour la date également pour un/des répertoire(s).

En tout cas vous êtes en mesure de créer des fichiers. Nous pouvons ensuite les manipuler.

`cp` (*copy*) : `cp src dest` permet de copier le fichier/répertoire `src` *en* `dest`. Si `dest` n'existait pas c'est le nom que portera la copie ; si `dest` existe et est un répertoire, le fichier `src` sera copié en un fichier du même nom placé dans le répertoire `dest` ; si `dest` est un fichier régulier existant et si `src` est un fichier régulier, le fichier `dest` est remplacé par une copie de `src` (à moins d'utiliser l'option `-i` (*interactive*).
On ne peut pas utiliser le commande `cp` sur un répertoire non vide ; pour copier un répertoire non vide, il faut utiliser l'option `-r` (*recursive*).

`mv` (*move*) : s'utilise comme `cp`, mais pour déplacer un fichier/répertoire

- si le deuxième paramètre existe et est un répertoire, le déplacement a lieu vers ce lieu
- s'il n'existe, la commande effectue un renommage avec ce nouveau nom

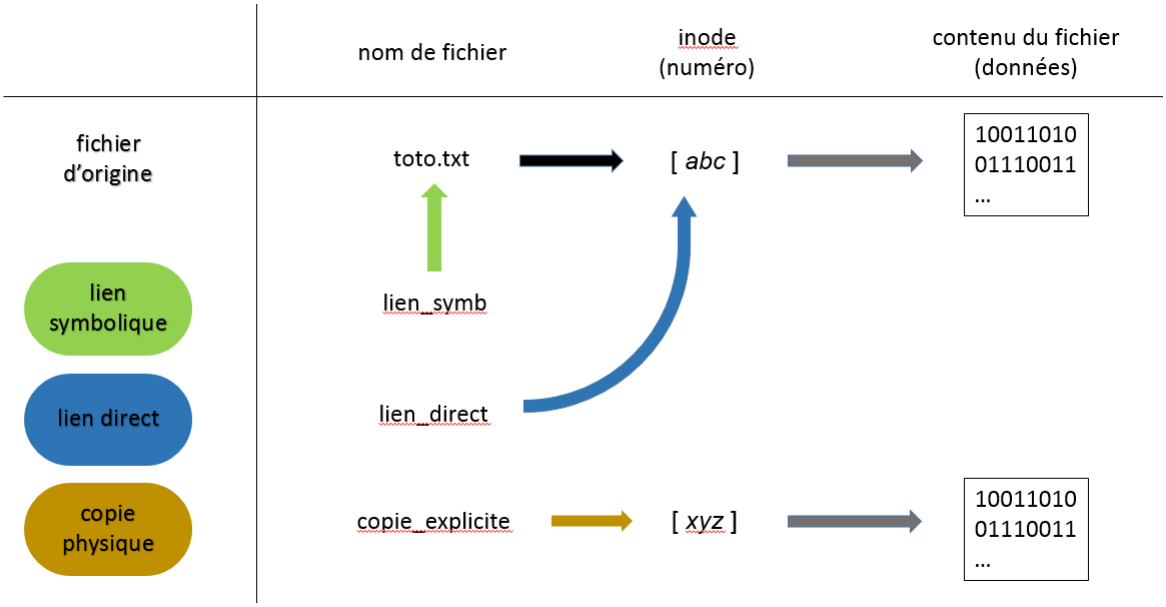
`rm` (*remove*) : supprimer un fichier (principales options : `-i`, `-f`, `-r`)
Cette commande n'est pas prévue pour les répertoires, mais on peut l'utiliser tout de même sur un répertoire : avec l'option `-r` on peut supprimer un répertoire non vide.

La commande `ln` (*link*) permet de faire des copies par lien. Il existe deux types de lien :

- lien *direct*, lien *physique*, ou lien *dur* (*hard link*)
- lien *symbolique* (avec l'option `-s`)

5. disponible sous `bash` uniquement
6. en particulier la pile n'est jamais vide : au minimum elle contient le nom du répertoire courant

Pour comprendre la différence, précisons que dans les systèmes Unix les fichiers [et répertoires] ont un **inode** (index de nœud : *node index*), numéro d'identification qui permet d'accéder aux données qu'il contient. La figure ci-dessous présente la différente entre lien direct, lien symbolique et copie physique ; la figure suivante (page suivante ?) donne un exemple complet.



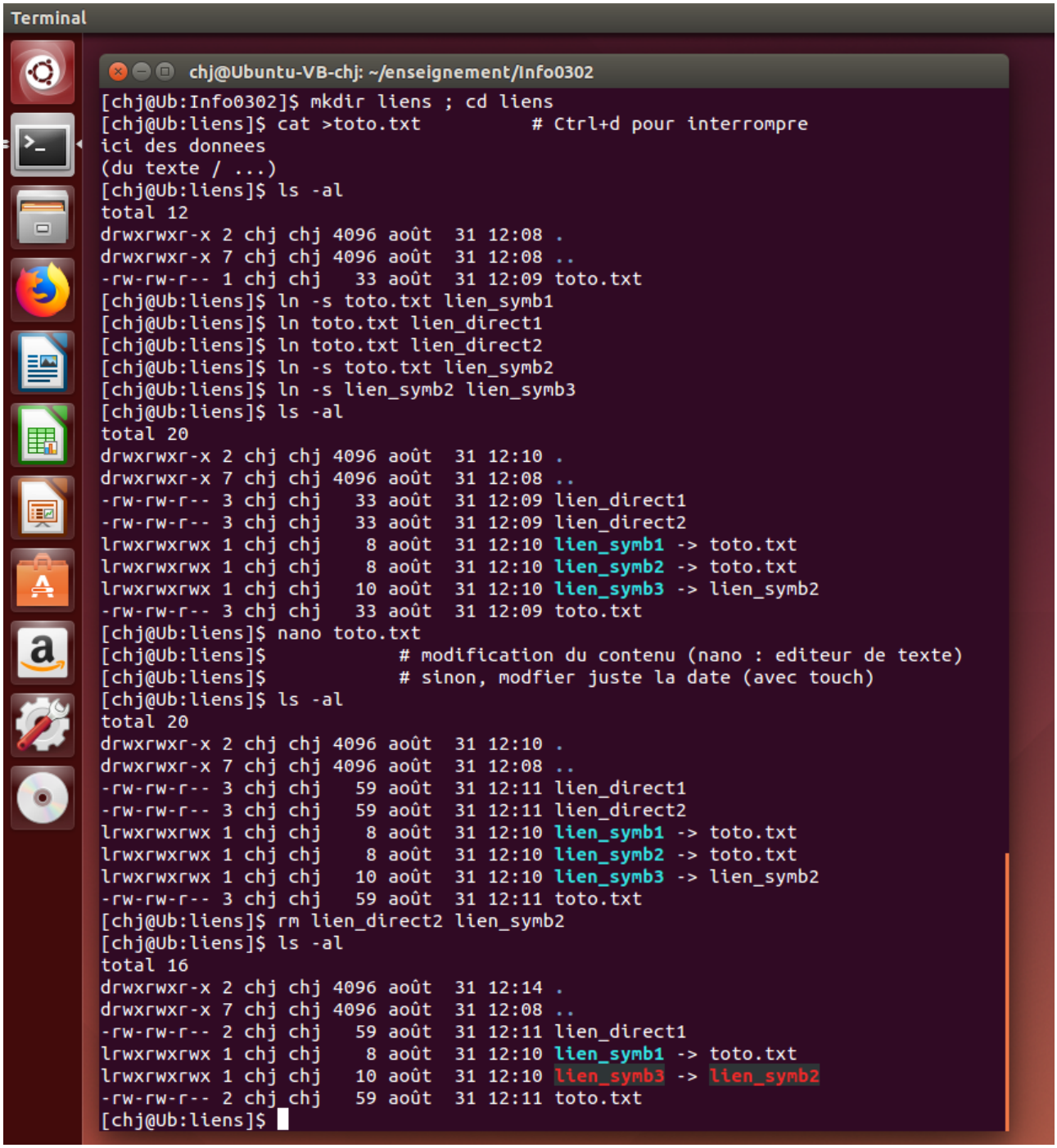
à noter : - on ne peut pas faire de lien direct sur un répertoire
- on parle de lien direct *sur* un fichier [régulier] et de lien symbolique *vers* un fichier

2.2.4 Combiner plusieurs commandes

Grouper des commandes

commande1 ; ***commande2*** **:** lance *commande1* puis *commande2*
commande1 **&&** ***commande2*** **:** (ET) ne lance *commande2* que si *commande1* a réussi
commande1 **||** ***commande2*** **:** (SINON) lance *commande2* si *commande1* a échoué

```
[chj@UB:Info0302]$ ls
codes/       liens/       toto.txt
[chj@UB:Info0302]$ ls toto.txt ; cat toto.txt
toto.txt
ici des donnees
(du texte / ...)
[chj@UB:Info0302]$ ls tata.txt ; cat tata.txt      # ko mais continue
ls: impossible d'accéder à tata.txt: Aucun fichier ou dossier de ce type
cat: tata.txt: Aucun fichier ou dossier de ce type
[chj@UB:Info0302]$ ls tata.txt && cat tata.txt     # ko : arret
ls: impossible d'accéder à tata.txt: Aucun fichier ou dossier de ce type
[chj@UB:Info0302]$ ls tata.txt || echo "probleme"  # ko : commande alternative
ls: impossible d'accéder à tata.txt: Aucun fichier ou dossier de ce type
probleme
```



liens (ln) : liens symboliques / liens directs

à noter : les structures de contrôle du shell offrent d'autres possibilités (mots clés `if`, `for`, ...)
=> voir partie suivante

Redirections

Les commandes utilisent des données et produisent des résultats.

Prenons l'exemple d'une commande division qui prend deux valeurs et en calcule le quotient (quotient de la première valeur par la deuxième). Les données peuvent être récupérées de différentes manières :

- sur la ligne de commande

- dans un fichier
- par des saisies au clavier

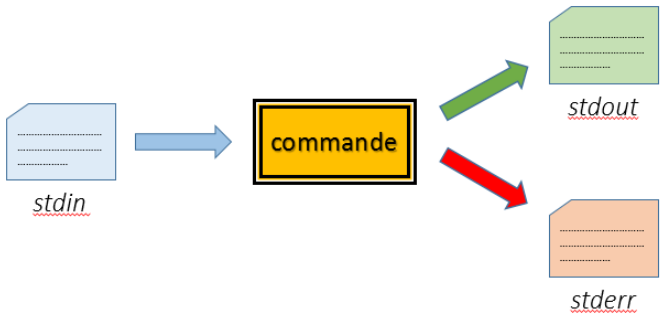
```
[chj@Ub:codes]$ ./div1 15 6
2
[chj@Ub:Info0302]$ ./div2 valeurs.txt
2
[chj@Ub:Info0302]$ ./div3
15
6
2
[chj@Ub:Info0302]$ ./div3
15
0
./div3: ligne 4: let: z=15/0 : division par 0 (le symbole erroné est "0")
[chj@Ub:Info0302]$ _
```

De même les résultats peuvent être produits à l'écran, enregistrés dans un fichier, ...
Par exemple,

- la commande `ls` prend comme donnée une ou plusieurs chaîne(s) de caractères issue(s) de la ligne de commande et produit son résultat par un affichage à l'écran
- la commande `div3` de l'exemple précédent prend ses données au clavier et envoie son résultat à l'écran : on dit que son **entrée** est le clavier et que sa **sortie** est l'écran.

Pour distinguer les sorties qui correspondent à des résultats normaux de ceux correspondant à des erreurs (exemple de la division par 0), on prévoit de distinguer la **sortie** de la **sortie d'erreur**. Par exemple, dans le cas de la commande `div3`, la division de 15 par 6 ne provoque pas d'erreur donc la valeur 2 est envoyée sur la sortie standard ; la division de 15 par 0 provoque une erreur donc son résultat (message d'erreur) serait plutôt envoyé sur la sortie d'erreur.

Toute commande a une **entrée standard**, une **sortie standard** et une **sortie d'erreurs standard**. Elles sont nommées respectivement *stdin*, *stdout*, *stderr*. Pour toutes les commandes, par défaut, l'entrée standard est le clavier alors que la sortie standard est l'écran et la sortie d'erreurs standard est l'écran.



Rappelons que dans les systèmes Unix tout est fichier. Donc *stdin*, *stdout* et *stderr* sont des fichiers. Pour chacun de ces fichiers une **redirection** permet de remplacer le fichier par défaut par un fichier régulier de votre choix. Par exemple *brancher* l'entrée standard d'une commande sur un fichier permet d'en automatiser l'utilisation pour éviter à l'utilisateur de faire des saisies fastidieuses à la main : la syntaxe est *commande* < *fichier*.

cmd < *ficin*	redirection de l'entrée standard
cmd > *ficout*	redirection de la sortie standard (s'il existe, *ficout* est écrasé)
cmd >> *ficout*	redirection de la sortie standard (s'il existe, *ficout* est continué)
cmd 2> *ficerr*	redirection de la sortie d'erreurs standard
cmd 2>> *ficerr*	redirection de la sortie d'erreurs standard *ficerr* complété)

à noter : pour faire *disparaître* les erreurs il suffit de brancher la sortie d'erreurs standard sur le fichier virtuel `/dev/null`

Pipeling

Un *pipeline* consiste à brancher l'entrée standard d'une commande sur la sortie standard d'une autre commande : la sortie de la première est reprise comme entrée de la seconde : *cmd1* | *cmd2*
Par exemple on peut compter le nombre d'entrées d'un répertoire en enchaînant la commande `ls` et la commande `wc` (*word count*), dont l'option `-w` permet de compter les mots.

```
[chj@Ub:codes]$ ls | wc -w
```

NB : La commande `tee` permet de dupliquer son entrée standard sur un fichier et sur sa sortie standard. Ainsi pour conserver la sortie standard d'une commande dans un fichier et en même temps l'utiliser en entrée d'une autre commade on intercale l'utilisation de la commande `tee` :
cmd1 | `tee` *fichier* | *cmd2*

```
[chj@Ub:codes]$ ls | tee liste | wc -w ; echo " entrees : " ; cat liste
```

à noter : lorsqu'on veut utiliser le résultat d'une commande non pas comme entrée d'une autre mais de façon plus "libre", on peut utiliser des *anti-quotes* (‘) **caractères d'échappement**.

```
[chj@Ub:codes]$ echo "Le repertoire courant contient ‘ls | wc -w‘ entrees"
```

Une combinaison de ces deux possibilités permet d'obtenir un résultat convenable :

```
[chj@Ub:codes]$ echo "Le repertoire courant possede ‘ls | tee fic | wc -w‘ entr
ees : " ; cat fic        # cette ligne de commande s'affiche sur deux lignes =>
Le repertoire courant possede 4 entrees :
div
err.txt
fic
out.txt
```

2.2.5 Filtres d'affichage des fichiers

Lorsqu'on considère les fichiers réguliers pour leur contenu, on les parcourt comme des **flux**, qu'on parcourt séquentiellement. On peut alors appliquer des **filtres** sur ces flux, pour réaliser un traitement

en cours de parcours : affichage simple des données, affichage interrompu, tri, ...[7]

Il existe plusieurs catégories de filtres : d'affichage (`cat`, `more`, `head`, `tail`), d'analyse de flux (`wc`), de recherche (`grep`), de tri (`sort`), ...

Filtres d'affichage

`cat` : affichage

- par défaut, renvoie le flux d'entrée stadard (*stdin*) sur la sortie standard (`stdout`)
  Ca semble inutile (essayez !) ; c'est avec des paramètres et des redirections que c'est intéressant.
- affichage simple : `cat` *`fichier`*
- concaténation de fichiers : `cat` $fichier_1$ ... $fichier_n$
  Les fichiers sont envoyés vers *stdout*, dans l'ordre et sans séparation.
  Pour concaténer vers un fichier, rediriger la sortie vers un fichier : `cat toto*.txt >tous`
- création d'un fichier : rediriger le clavier (*stdin*) vers un fichier :

```
\item création de fichier : cat > fichier
mon texte.
^D
```

`more` (ou `less`) : affichage page par page

- consultation d'un fichier : `more` *`fichier`*
  Pour la navigation : comme avec `man` => touches *espace*, *entrée* et `q` (*`/expr`* pour lancer une recherche)
- consultation page à page du résultat d'une commande : *`commande`* `| more`

`head` : affichage du début du flux d'entrée

- `-n` n : affichage des n premières lignes (ou `-`n)
- `-c` n : affichage des n premiers caractères

`tail` : affichage de la fin du flux d'entrée

- `-n` n : affichage des n dernières lignes (ou `-`n)
- `-c` n : affichage des n derniers caractères
- `+`n : affichage de la fin du fichier en commençant à la n-ième ligne

`wc` : comptage des lignes, mots et caractères[8]

- `-c` pour le nombre de caractères
- `-w` pour le nombre de mots
- `-l` pour le nombre de lignes
- sans option ? équivalent à `-lwc`

7. Il s'agit de traitements sur les flux, pas sur les données.
8. bien qu'il ne s'agisse pas d'un filtre

Gestion des champs

`cut` : extraction et réorganisation de champs (pour un fichier présenté en colonnes)

- `-d c` : utiliser le caractère c comme séparateur de champs (par défaut, la tabulation)
- `-f liste` : coupure en mode champ (`-c liste` pour couper en mode caractère)

  Format de la liste :

n	le n-ième
m`,`n	le m-ième et le n-ième
m`-`n	du m-ième au n-ième
n`-`	du n-ième à la fin
`-`n	du 1er au n-ième

```
[chj@Ub:Info0302]$ cat toto.txt
1   2   3   4   5   6
ab  cd  ef  gh  ij  kl
[chj@Ub:Info0302]$ cut -f 5,2 toto.txt
2   5
cd  ij
[chj@Ub:Info0302]$ cut -c 2-4,7-8 toto.txt
    2   4
b   c
```

Recherche dans un flux

`grep` : recherche d'expression dans un flux (affichage des lignes correspondant au critère)

1. recherche d'une expression simple (chaîne de caractère)

   définition du critère de recherche : passé en paramètre

   options :

`-n`	préfixe les lignes affichées par leur numéro de ligne dans le fichier (numéro initial)
`-c`	compte les lignes répondant au critère
`-l`	avec le nom du/des fichier(s) répondant au critère (en cas de recherche dans une liste de fichiers)
...	
`-i`	ignore la casse (distinction minuscules/majuscules)
`-v`	inversion du critère de recherche
`-f` *fichier*	l'expression est dans le fichier dont le nom est spécifié avec cette option

2. précisions sur l'expression cherchée :
   pour préciser si l'expression cherchée est en début ou en fin de ligne, ...

`^`	le caratères *début de ligne* => `grep ^bon fichier`
`$`	le caractère *fin de ligne* => `grep jour$ fichier`
`"`*mot*`"`	le mot exactement (éviter bonjour, ...)
`'`*chaine*`'`	la chaîne peut contenir des espaces, ...

3. recherche complexe (expression régulière) L'expression peut être générique, grâce à l'utilisation de méta-caractères, de groupements, de répétitions :
   - les caractères `^` et `$` utilisés ci-dessus sont des méta-caractères donc `jour$` est une expression régulière de `grep`

- tout caractère qui n'est pas un méta-caractère peut être utilisé dans une expression régulière
- le caractère \ peut être utilisé comme caractère d'echappement afin d'utiliser explicitement un méta-caractère en tant que caractère => `jour\$` pour considérer l'expression `jour$`
- `.` désigne un caractère quelconque (qui apparaît une fois)
- `[abc]` désigne un des trois caractères `a`, `b` ou `c`
- `[^abc]` désigne un caractère qui ne soit pas un `a`, un `b` ou un `c`
- `C-E` désigne la plage de caractères de `C` à `E`
- `[abc]` désigne un des trois caractères `a`, `b` ou `c`
- `[a-z]` : une lettre minuscule
- `[^a-z]` : un caractère qui ne soit pas une lettre minuscule
- `*` : répétition, un nombre quelconque de fois
- `+` : répétition, au moins une fois
- `{`*n*`}` : répétition, n fois exactement
- ...
- suite d'expressions (et) : $expr_1 expr_2$
- expression alternative (ou) : $expr_1$`|`$expr_2$
- *marquer* une expression : `(`***expr***`)`

Par exemple, un numéro de téléphone français, suite de 10 chiifres commençant par un 0 mais pas par deux 0, sera décrit par l'expression régulière `0[1-9][0-9]{8}`

Filtre de tri

`sort` : tri ligne par ligne, par ordre croissant

- `-n` : tri numérique (`-g` pour les réels)
- `-r` : inverse l'ordre du tri
- `-f` : ignore la casse
- `-t` *c* : utilise le caractère c comme séparateur de champs dans la ligne (par défaut, l'espace)
- `-k` ***fields*** : tri en fonction du sélecteur de champ

  principe de sélection des champs (*fields*) :

n	tri selon la fin de la ligne, en commençant à partir du n-ième champ
m`,`*n*	tri selon les champs à partir du champ m et jusqu'au champ n
n`,`*n*	tri selon le n-ième champ exactement
n`.`*c*	selon la fin de la ligne, à partir du c-ième caractère du n-ième champ
`m.x,n.y`	...

  exemple : si une ligne du fichier à trier est la suivante : `toto titi ta:ta` alors :

  - si l'option `-t` n'est pas précisée, il y a 3 champs : `toto`, `titi` et `ta:ta`
  - avec l'option `-t :`, il y a deux champs : `toto titi ta` et `ta`
  - l'option `-k 2,2` permet de trier sur le deuxième champ seulement (`titi`)
  - l'option `-k 2.2` tri à partir du deuxième caractère du deuxième champ (`ti ta:ta`)

`uniq` : gestion des entrées multiples (supprime les entrées multiples consécutives)

- `-u` : affiche seulement les lignes uniques
- `-c` : affiche le nombre d'occurrences
- `-d` : affiche seulement les lignes multiples
- `-i` : ignore la casse

Autres

Il existe de nombreux autres filtres permettant d'accomplir des actions évoluées : voir le support mis à disposition par P Mignot pour l'étude de `od`, `tr`, `sed` ou `join`, et des exemples complémentaires d'expressions régulières pour `grep`.

Bien que ce ne soit pas un filtre mentionnons la commande `diff`, qui permet de mettre en évidence toutes les différences entre deux documents : dans le cas de versions différentes d'un même document, cette commande vous permettra de localiser exactement les évolutions entre les deux versions.

```
[chj@Ub:codes]$ man diff >diff-man.txt
[chj@Ub:codes]$ info diff >diff-info.txt
[chj@Ub:codes]$ ls -l
total 20
-rw-rw-r- 1 chj chj 10534 sept.  2 21:12 diff-info.txt
-rw-rw-r- 1 chj chj  6585 sept.  2 21:12 diff-man.txt
[chj@Ub:codes]$ diff diff-info.txt diff-man.txt > diff-diff
[chj@Ub:codes]$ wc -l diff-diff
561 diff-diff
[chj@Ub:codes]$ ls -l
total 40
-rw-rw-r- 1 chj chj 18263 sept.  2 21:13 diff-diff
-rw-rw-r- 1 chj chj 10534 sept.  2 21:12 diff-info.txt
-rw-rw-r- 1 chj chj  6585 sept.  2 21:12 diff-man.txt
[chj@Ub:codes]$ more diff-diff
1,2c1
< /usr/share/info/diffutils.info.gz
< File: diffutils.info-t,  Node: Invoking diff,  Next: Invoking diff3,  Prev: I
nvoking cmp,  Up: Top
--
> DIFF(1)                          User Commands                          DIFF(1)
4,5d2
< 13 Invoking ‘diff’
< ******************
7d3
< The format for running the ‘diff’ command is:
-Plus-(3%)
```

2.2.6 Recherche de fichiers

Les commandes `which` (commande) et `locate` (fichier indexé)

Pour localiser la fichier d'une commande, on utilise la commande `which`.

La commande `locate` permet de faire une recherche simple de fichier (ou répertoire/...). Elle s'appuie sur une base de données des fichiers du système, mise à jour régulièrement. Il se peut donc qu'elle ne soit pas exactement à jour, et donc qu'elle ne permette pas de repérer un fichier créé ou déplacé récemment ; vous pouvez mettre à jour la base de données d'indexation en invoquant la commande `updatedb` [en mode super-utilisateur].[9]

```
[chj@Ub:Info0302] which ls
/bin/ls
[chj@Ub:Info0302] locate div-chj
[chj@Ub:Info0302] sudo updatedb
password for chj:
[chj@Ub:Info0302] locate div-chj
/home/chj/enseignement/Info0302/codes/div-chj1
/home/chj/enseignement/Info0302/codes/div-chj2
/home/chj/enseignement/Info0302/codes/div-chj3
```

La commande `find`

Contrairement à `locate`, `find` ne s'appuie pas sur une base de données : à chaque requête elle parcourt la totalité du système de fichier poour réaliser la recherche.

`find` : recherche et exécution de commandes récursivement sur une arborescence[10]

syntaxe générale : `find` *`chemin critère action`*

- *chemin* : répertoire initial de la recherche (on cherche dans tout ce répertoire)
- *critère* : critère [éventuel] sur le nom du fichier, son type, sa taille, ses droits, la date d'accès ou de modification, ...
- *action* : pour chaque fichier correspondant aux critères on réalise une action : affichage du chemin, affichage du contenu (pour un répertoire), exécution d'une commande

notations :

- pour les noms de fichiers, les chaînes de caractères sont placées entre doubles quotes (") et les expressions régulières *regexf* entre simples quotes (')
- arguments numériques :
  - *`val`* la valeur *val*, exactement
  - *`+val`* valeur au moins égale à *val*
  - *`-val`* valeur au plus égale à *val*

9. `locate` est efficace car elle s'appuie sur une base de données d'indexation ; en contre-partie, la reconstruction [de la totalité] de la base de données est longue.
10. il ne s'agit pas d'un filtre

```
[chj@Ub:codes]$ find /etc -name passwd
find: «/etc/cups/ssl»: Permission non accordée
/etc/cron.daily/passwd
/etc/passwd
find: «/etc/ssl/private»: Permission non accordée
find: «/etc/polkit-1/localauthority»: Permission non accordée
/etc/pam.d/passwd
[chj@Ub:codes]$ find /etc -name passwd 2>/dev/null
/etc/cron.daily/passwd
/etc/passwd
/etc/pam.d/passwd
[chj@Ub:codes]$ find / -name passwd 2>/dev/null | wc -l
7
[chj@Ub:codes]$ find / -name passwd -exec wc -l '{}' \; 2>/dev/null
9 /etc/cron.daily/passwd
36 /etc/passwd
6 /etc/pam.d/passwd
172 /usr/bin/passwd
0 /usr/share/doc/passwd
23 /usr/share/bash-completion/completions/passwd
6 /usr/share/lintian/overrides/passwd
[chj@Ub:codes]$ find / -type f -size +100M 2>/dev/null | wc -l
2
[chj@Ub:codes]$
```

critères : (liste non exhaustive)

`-name` *nom*	nom du fichier (ou expression régulière) ; `iname` *xx* : ignorer la casse
`-path` *chemin*	... (ou `-ipath` *chemin*)
`-size` *n*	...
`-empty`	...
`-type` *c*	type du fichier => types possibles : `d` (répertoire), `f` (fichier), `l` (lien)
`-perm` n	les droits du fichier (en octal)
`-atime` *nbjours*	accès il y a *nbjours* jours
`-amin` *nbmin*	accès il y a *nbmin* minutes
`-mmin` *nbmin*	contenu modifié il y a *nbmin* minutes

action : sur chaque fichier trouvé correspondant aux critères

`-print`	affichage [du chemin] (action par défaut)
`-fprint` *fichier*	idem, mais sauvegarde les chemins dans un fichier
`-ls`	(pour un répertoire) affichage du contenu ; `-fls fichier` : idem + sauvegarde
`-exec` *commande* `{};`	exécute la commande sur chaque fichier/répertoire correspondant `{}` désigne le nom du fichier ou répertoire courant (sur lequel la commande est exécutée)

2.2.7 Archivage

tar

`tar` permet d'archiver/désarchiver des fichiers.
Il y a plusieurs modes, et des options supplémentaires :

mode (un mode, au choix)

- `c` : création
- `x` : extraction
- `t` : contenu
- `r` : ajout

options supplémentaires (éventuellement)

- `v` : *verbose* => annonce les opérations réalisées)
- `f` : *file* => sortie dans un fichier plutôt que sur un disque
- `z` : *zip* => archiver à la volée
- `r` : ajout

extensions (usage)

- `.tar` : archives
- `.tar.gz` : archives, compressées avec gzip dans un second temps
- `.tgz` : archives compressées (à la volée)

gzip / gunzip : compression/décompression uniquement (pas archivage)
`gzip man-diff.txt` : compression => crée man-diff.txt.gz
`gzip /etc/passwd man-diff.txt.gz` : décompresse
`gzip -d man-diff.txt.gz` : décompresse

```
[chj@Ub:~]$ tar vcf sv3.tar enseignement/Info03*
        ...
[chj@Ub:~]$ gzip sv3.tar
        ...
[chj@Ub:~]$ gunzip sv3.tar.gz
        ...
[chj@Ub:~]$ tar xf sv3.tar
        ...
```

```
[chj@Ub:~]$ tar zcvf sv3.tgz enseignement/Info03*
        ...
[chj@Ub:~]$ tar xvf sv3.tgz
        ...
```

voir aussi...

compress / uncompress (fichiers compressés au format `.Z`)

`zip / unzip` : archives compressées (au format `.zip`)
compresser un répertoire : `zip -r toto.zip /enseignement/Info0302/`
décompresser une archive : `unzip toto.zip`

2.3 Utilisateurs et droits

Les systèmes Unix sont multi-utilisateurs. Les diférents utilisateurs sont organisés par groupes (ensembles d'utilisateurs ayant les mêmes types de possibilités).
Par exemple si on envisage les groupes `profs` et `etudiantLicence`, on peut imaginer que les profs auront tous le même quota d'espace disque et accès à certains répertoires partagés en écriture, alors que les étudiants auront un quota disque moins important et auront un accès au répertoire partagé, mais uniquement en lecture.

L'utilisateur `root`

Le super-utilisateur, **root**, peut voir et modifier les fichiers des utilisateurs ; c'est lui qui a la possibilité de gérer les groupes et les utilisateurs.
Son groupe est `root`. Il a tous les droits sur l'ensemble des fichiers du système.

```
root@Ub:~>% cd ../chj/Info0302/
root@Ub:/home/chj/Info0302>% ls
codes/  liens/  toto.txt
root@Ub:/home/chj/Info0302>% cd
root@Ub:~>%
```

```
root@Ub:~>% addgroup etudiants
        ...
root@Ub:~>% adduser kevin      # user kevin ; group kevin
root@Ub:~>% adduser xx         # user xx ; group xx
root@Ub:~>% deluser xx         # ...
root@Ub:~>% addgroup etudiantsLicence
        ...
root@Ub:~>% delpgroup etudiants
        ...
root@Ub:~>% usermod -g etudiantsLicence kevin        \
                               # ajoute au groupe (changement de groupe)
root@Ub:~>% ...
```

Les utilisateurs standards

Les utilisateurs standards ont certains droits d'administration sur leur propre compte, notamment

- la modification du mot de passe (commande `passwd` ou `yppasswd`)
- la fermeture de session (commande `halt`)
- installer des programmes pour son propre compte (dans son compte) ... si le système l'y autorise.
- ...

Sur un système de type **Desktop** vous êtes à la fois utilisateur et super-utilisateur, mais vous êtes connecté par défaut en tant qu'utilisateur standard :

- `sudo` permet de lancer une commande en *mode root* (montage de disque, ...).[11]
- `su` permet de passer durablement en mode root (jusqu'à déconnection : `Ctrl+d` ou `exit`)
  NB : il faut des droits élevés pour devenir root donc il faut taper la commande `sudo su`.

Propriétaire d'un fichier

Chaque fichier a un propriétaire et un groupe propriétaire. Ca fait partie des informations affichées par la commande `ls -l`.
L'utilisateur `root` peut changer le propriétaire ou le groupe d'un fichier (commandes `chown` et `chgroup`).

Les droits sur un fichier

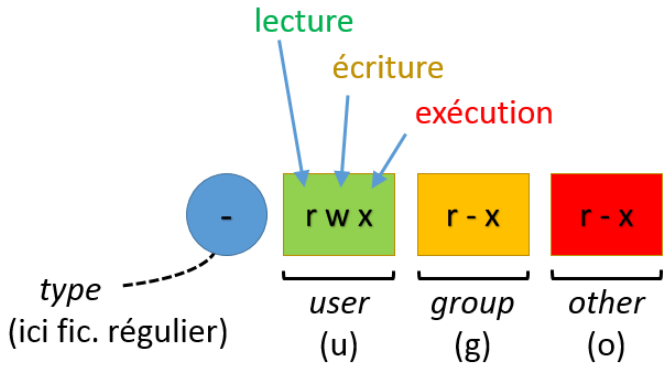
A part pour `root`, qui a tous les droits sur tous les fichiers on peut se les octroyer, chacun a des droits différents sur chaque fichier :

- le propriétaire a des droits particuliers sur ses fichiers (qui peuvent être élevés)
- les autres membres du groupe de l'utilisateur peuvent aussi avoir des droits dessus
- enfin les membres d'autres groupes ont des droits encore moindre sur ce fichier.

Par ailleurs il existe trois types de droits :

- *lecture* (**r** : *Read*) => consultation
- *écriture* (**w** : *Write*) => modification
- *exécution* (**x** : *eXecute*) => ce droit n'est utile que pour les fichiers de programmes et de scripts

La première colonne affichée par `ls -l` donne le type du fichier (voir 1.3.2) puis 9 caractères qui correspondent aux droits, activés ou non.



lecture	+	+	+
écriture	+	-	-
exécution	+	+	+
-	r w x	r – x	r – x
	user (u)	*group* (g)	*other* (o)

Changer les droits d'accès à un fichier : `chmod`

Le propriétaire d'un fichier peut en changer les droits grâce à la commande `chmod`, qui s'utilise de deux façons différentes :

11. Bien sûr il faut connaître le mot de passe de root!!

`chmod` ***relatif*** : pour modifier les droits d'un type d'utilisateur on peut ...
... ajouter ou supprimer (`+` / `-`) des droits ...
... de lecture, d'écriture ou d'exécution (`r` / `w` / `x`) ...
... au user, ou groupe ou aux autres (`u` / `g` / `o`).
=> Par exemple

- pour ajouter (`+`) au groupe (`g`) le droit d'écriture (`w`) sur un fichier *fic*, l'utilisateur exécute la commande `chmod g+w fic`
- pour enlever (`-`) aux autres (`o`) le droit d'exécution (`x`) sur un fichier *fic*, l'utilisateur exécute la commande `chmod o-x fic`

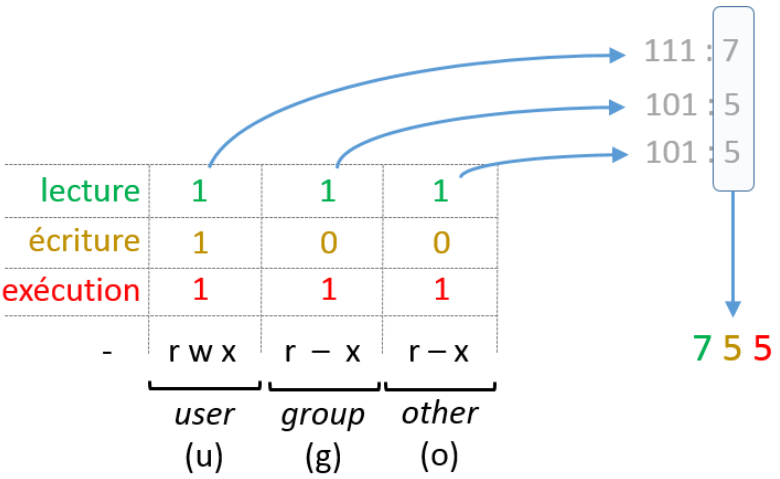
```
[chj@Ub:Info0302]$ chmod g+wo-x toto.txt
        ...
```

`chmod` ***absolu*** : la seconde figure présente les droits par bloc :

- les droits de l'utilisateur sont `rwx` : `111` en binaire donc 7 en octal
- les droits du groupe sont `r-x` : `101` en binaire donc 5 en octal
- les droits des autres sont `r-x` : `101` en binaire donc 5 en octal

=> ainsi globalement, en octal, les droits du fichier sont 755.

Si on souhaite ajouter à ce fichier les droits d'écriture pour le groupe et supprimer le droit d'exécution pour les autres, les droits doivent passer à `rwx rwx r-`, donc `774` en octal :
la commande est alors `chmod 774 fic`




```
[chj@Ub:Info0302]$ chmod 774 toto.txt
        ...
```

Nous verrons dans le deuxième partie qu'un script shell est écrit dans un fichier texte. Il faut ensuite que l'utilisateur attribue le droit d'exécution à ce fichier, et aisi on peut l'évoquer comme une commande.

2.4 Gestion des processus

Un processus Unix est une exécution d'un programme / d'une commande. On peut avoir plusieurs processus pour une même commande (par exemple plusieurs exécutions smultanées de la commande `bash`).

2.4.1 Les caractéristiques des processus

Eléments caractéristiques :

- son `PID` (*Process ID*) : numéro, unique, affecté lors de son lancement
- son `PPID` : PID du processus père (qui l'a lancé)
- le propriétaire (l'utilisateur qui a lancé la commande)
- sa priorité, de -20 (minimale) à 20 (maximale) [priorité normale : 0]
- ses caractéristiques d'exécution
  - caractéristiques mémoire : mémoire vive, mémoire virtuelle
  - caractéristiques temporelles : date de début, temps CPU, ...
- son état (STATUS) : voir ci-dessous

Informations sur les processus : les commandes

- `ps` : état ponctuel des processus en cours d'exécution
- `top` : état en continu (en haut les processus utilisant le plus de ressources) (`q` pour quitter)
- `time` : état *a posteriori*
  - on lance la commande avec `time` *`commande`*
  - à la fin on obtient des informations : temps réel, temps utilisateur, temps système
- voir aussi : `pstree` : affiche l'arbre de parenté des processus

Seul le propriétaire d'un processus [ou root] peut le contrôler.

2.4.2 *jobs* en premier plan / en arrière-plan

Un processus lancé par un `shell` est un `job`. Il peut être lancé en premier plan ou en arrière plan, et être passé "à la main" de premier plan à arrière plan, ou l'inverse.

Premier plan / arrière plan :

- un job lancé au premier plan ne se détache pas du processus qui l'a lancé : on doit attendre la fin de son exécution pour "reprendre la main"
- lorsqu'on lance un job en arrière plan (en tâche de fond) on récupère la main tout de suite : le job s'exécute en parallèle du shell qui l'a lancé

Commandes de gestion des jobs :

- *`commande`* : lancement de la commande [en premier plan]
- *`commande`* `&` : lancement de la commande en arrière plan

- `jobs` : liste des jobs en cours d'exécution
  avec les `jid` (*job ID*) : numéro de job
- `bg %jid` (*background*) : passe à l'arrière plan (`%jid` : le job dont le `jid` est spécifié
- `fg %jid` (*foreground*) : passe au premier plan
- `Ctrl+c` : tue le job en premier plan
- `Ctrl+z` : suspend le job en premier plan (`bg` le basculera alors à l'arrière plan)
- NB : `nohup commande &` : lancement en tâche de fond et même **détâchée** ; précise que la commande est indépendante de son père (ne doit pas se terminer si son père meurt)

2.4.3 Etats d'un processus

Un processus peut être dans différents états à un moment donné (généralise la notion de job ; plus précis que arrière-plan / premier plan).

Sous Unux les différents états sont :

- `R` (*running*) : en exécution
- `S` (*sleeping*) : endormi
- `T` (*stopped*) : stoppé
- `Z` : zombie
- `D` (*dead*) : sommeil définitif

Le passage entre les différents états se réalise par l'envoi de **signaux**, ou **interruptions**.

Envoyer un signal : `kill -num pid`
(par défaut *num* vaut 15)

Les principaux signaux :

2	SIGINT	interruption
3	SIGQUIT	interruption avec *core* (sauvegarde de l'état dans un fichier)
9	SIGKILL	terminaison forcée
15	SIGTERM	terminer (défaut)
17	SIGSTOP	stopper (*pause*)

NB : - un processus père ne peut pas mourir avant la mort de tous ses processus fils
- la fin du processus père entraine la fin de tous ses processus fils (sauf lancement avec `nohup`)
- commande `wait` : rend la main lorsque tous les processus en tâche de fond sont terminés

2.4.4 Autres commandes

`at batch` : lancement différé (à une certaine date)

`nice` / `renice` (root) : changer la priorité d'un processus

2.4.5 Exemple

```
[chj@Ub:Info0302]$ top
top - 10:21:02 up 1 day, 11:34,  3 users,  load average: 0,22, 0,32, 0,20
Tâches: 167 total,   1 en cours, 166 en veille,   0 arrêté,   0 zombie
%Cpu(s):  3,3 ut,  7,0 sy,  0,0 ni, 89,7 id,  0,0 wa,  0,0 hi,  0,0 si,  0,0 st
KiB Mem:   2068068 total,  1391204 used,   676864 free,   122828 buffers
KiB Swap:  2095100 total,        0 used,  2095100 free.   625172 cached Mem

  PID UTIL.     PR  NI    VIRT    RES    SHR S %CPU %MEM    TEMPS+ COM.
 3682 chj       20   0  355620 159292  60068 S  4,6  7,7  41:15.02 compiz
 1038 root      20   0  232764  66668  29704 S  1,6  3,2  32:51.21 Xorg
 1214 chj       20   0   18032   3116   2784 S  1,6  0,2  31:50.24 VBoxClient
 4512 chj       20   0    5532   2872   2472 R  1,6  0,1   0:00.29 top
 1345 chj       20   0  139512  30396  21076 S  0,7  1,5   0:52.56 unity-panel+
   27 root      20   0       0      0      0 S  0,3  0,0   1:36.69 kworker/0:1
[chj@Ub:Info0302]$ time ls
codes/    liens/     toto.txt

real        0m0.047s
user        0m0.004s
sys        0m0.012s
[chj@Ub:Info0302]$ find / -name "toto" 2>/dev/null
^Z
[1]+  Arrêté                 find / -name "toto" 2> /dev/null
[chj@Ub:Info0302]$ ps
  PID TTY          TIME CMD
 2483 pts/7    00:00:03 bash
 4514 pts/7    00:00:00 find
 4515 pts/7    00:00:00 ps
[chj@Ub:Info0302]$ bg %1
[1]+ find / -name "toto" 2> /dev/null &
[chj@Ub:Info0302]$ ps
  PID TTY          TIME CMD
 2483 pts/7    00:00:03 bash
 4516 pts/7    00:00:00 ps
[1]+  Termine 1               find / -name "toto" 2> /dev/null
[chj@Ub:Info0302]$ ps -u chj | grep bash
 2235 pts/1    00:00:02 bash
 2483 pts/7    00:00:03 bash
[chj@Ub:Info0302]$ kill -9 2235
[chj@Ub:Info0302]$ ps
  PID TTY          TIME CMD
 2483 pts/7    00:00:03 bash
 4640 pts/7    00:00:00 ps
[chj@Ub:Info0302]$ kill -9 4640
bash: kill: (4640) - Aucun processus de ce type
[chj@Ub:Info0302]$ _
```

2.5 Paramétrez votre environnement de travail

Il est possible de paramétrer son environnement de travail : en exécutant des commandes, en créant de nouvelles commandes, en modifiant des fichiers de configuration, en modifiant des variables d'environnement.

L'essentiel du paramétrage de l'environnement est effectué [globalement] au démarrage du système et login/déllogin de chaque utilisateur. La configuration est réalisée par l'exécution de commandes, par le super-utilisateur (mode *root*) ou par l'utilisateur (mode *user*). Les commandes en question sont stockées dans des fichiers, qui sont chargés au démarrage / au login et au délogin.

Fichiers de démarrage et de fin de session

- `/etc/profile` : chargé au démarrage du système (par `root`)
- `~/.profile` : chargé au login (U)
- `~/.bashrc` : chargé à chaque lancement de nouveau shell (U)
  (par un clic en mode console / par l'exécution de la commande `bash`)
- `~/.bash_logout` : chargé à la sortie d'un shell (U)

Variables d'environnement (ou *variables système*)

Les variables d'environnement caractérisent les sessions des utilisateurs, et leurs valeurs peuvent être propres à chacun ou à chaque groupe d'utilisateurs.
Contrairement aux variables utilisateur qui sont crées à l'initiative de l'utilisateur, les variables système sont prédéfinies et leurs valeurs sont fixées par le système.
Les principales variables d'environnement sont `HOSTNAME`, `USER`, `HOME`, `PATH`, ..., `PS1`.

la gestion des variables utilise les commandes suivantes :

- `echo $VAR` : afficher la valeur de la variable VAR (U/E)
- `VAR=`*`valeur`* : modifier la valeur d'une variable existante ou créer une nouvelle varable (une nouvelle variable n'est pas chargée dans l'environnement)
- `export VAR` : charger une variable dans l'environnement
- `export VAR=`*`valeur`* : modifier + charger dans l'environnement

- `printenv` : afficher l'ensemble des variables d'environnement
- `printenv VAR` : afficher une variable d'environnement particulière (si elle fait partie de l'environnement) ; voir aussi `printenv | grep VAR`

Ayez la curiosité d'analyser les contenus des fichiers `/etc/profile`, `~/.profile`, `~/.bashrc` et `~/.bash_logout`. De même, afichez la liste des variables d'environnement et consultez les valeurs des principales.

Evitez de modifier le contenu des fichiers `~/.profile`, `~/.bashrc` et `~/.bash_logout`.
Si vous modifiez le contenu du fichier `~/.bashrc`, vous ne pourrez en voir l'effet qu'à l'ouverture d'un nouveau shell (et pour ce shell uniquement). pour que les changements prennent effet pour le shell en cours d'exécution, il faut recharger le fichier `~/.bashrc`, par la commande `source ~/.bashrc`

Conservez le contenu de `PS1` dans une variable utilisateur et changez en la valeur ; ouvrez un autre shell pour vérifier si la valeur a été changée globalement ou non ; mettez le modification dans le fichier `~/.bashrc` et rechargez ouvrez un nouveau shell ; ...

Création d'alias un alias est un nom synonyme pour une commande [12]

```
[chj@Ub:Info0302]$ alias ls2='ls ; echo "--" ; ls'
[chj@Ub:Info0302]$ ls2
codes/  liens/  toto.txt
--
codes/  liens/  toto.txt
[chj@Ub:Info0302]$ alias        # liste des alias et leurs significations
        ...
[chj@Ub:Info0302]$ alias | grep "alias l"
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls -color=auto'
alias ls2='ls ; echo "--" ; ls'
[chj@Ub:Info0302]$ alias ls2
alias ls2='ls ; echo "--" ; ls'
[chj@Ub:Info0302]$ unalias ls2
[chj@Ub:Info0302]$ ls2
ls1 : commande introuvable
[chj@Ub:Info0302]$
```

Si vous créez ou modifiez un alias il ne vaut que pour le shell où il a été créé. Pour l'exporter, ajouotez l'appel de la commande au fichier `~/.bashrc` et rechargez ce fichier.

2.6 Compléments

Les supports de M Mignot présentent des éléments complémentaires. Avec son accord, vous pouvez les utiliser librement. En particulier je vous recommande de profiter des deux premiers, qui abordent respectivement les commandes de base et les filtres.

12. il ne s'agit pas d'un lien vers un fichier

3 *La pratique, il n'y a que la pratique !*

A vous de jouer...

Exercice 1 (Login et premières commandes)
1°) Utilisez la commande `printenv HOME` pour "déterminer" votre répertoire de connexion.

2°) Comment déterminer le shell que vous utilisez ?

Exercice 2 (Manipulation de fichiers et de répertoires)
1°) Quels sont les fichiers ou répertoires déjà présents sur votre compte ? (cachés ou non)

2°) Recréez l'arborescence suivante :
```
~/tp1
~/tp1/essai
~/tp1/essai/rep1
~/tp1/essai/rep2
```

3°) Copiez le fichier `/etc/passwd` dans le répertoire `rep1`.
Même chose dans le répertoire `rep2` mais en utilisant l'option `-i`.

4°) Utilisez les différentes options de `ls`.

5°) Renommez le répertoire `rep2` en `rep3`.

6°) Déplacez le répertoire `rep3` vers le répertoire `/tp1`.

7°) Utilisez la fonction `du` (pensez à utiliser `man du`).

Exercice 3 (Droits)
1°) Changez les droits sur le répertoire `rep1` en `-x- --- ---`. Peut-on encore écrire dedans ? (pour créer rapidement un fichier, utilisez la commande `touch`).

2°) Continuez à manipuler les droits de ce répertoire pour voir les effets (écriture seule, lecture seule). Utilisez les différentes notations (`ugo` et octale)

3°) liens
a) Expérimentez `ln` et `ln -s` sur vos propres exemples.
b) Quel est l'intérêt des liens ?

Exercice 4 (Processus)
1°) Utilisez toutes les commandes issues du support permettant d'obtenir des informations sur les processus.

2°) Lancez la commande `emacs`. En utilisant les commandes appropriées, stoppez emacs (attention, stopper n'est pas tuer), puis relancez-le en tâche de fond.

3°) Tuez emacs avec la commande `kill`.

4°) Que se passe-t-il si vous tuez le processus associé à `bash` ?

Exercice 5 (Redirections)
1°) Créez un fichier `exo1` qui contient le manuel de la commande `ls`.

2°) Que fait la commande `cat exo1` ? Et `cat exo1 > /dev/null` ?

3°) Copiez le fichier `exo1` vers `exo2`. Puis compressez `exo2`.

4°) Examinez le fic hier binaire compressé avec la commande `od` : examinez le fichier en hexadécimal, puis en mode caractère nommés.

5°) Lancez la commande `ls /truc /bin`. Redirigez la sortie erreur dans le fichier `ls.err` et la sortie standard dans le fichier `ls.out`, en une seule commande.

6°) Que fait la commande `cat` ? Et `cat > fic` ? (tapez sur le clavier après avoir lancé ces commandes ; CTRL+D pour quitter).

Exercice 6 (Tube / *pipe*)
1°) Comparez les tailles des deux fichiers `exo1` et `exo2.gz` de l'exercice précédent avec la commande `wc`.

2°) Utilisez la commande `more` (ou `less`) sur les deux fichiers.

3°) Idem avec `head` et `tail` : utilisez les options pour n'afficher que les deux premières lignes, que les deux dernières, et celles entre pour 10 et 100.

Exercice 7 (Recherche de fichiers)
1°) Écrivez une commande qui compte le nombre total de fichiers dans votre répertoire.

2°) Écrivez une commande qui affiche les fichiers qui ont été modifiés depuis une heure.

3°) Écrivez une commande qui permet de trouver tous les fichiers postscript (`.ps`) et de les compresser (si nécessaire créez-en).

4°) Écrivez une commande qui permet de trouver tous les fichiers texte (`.txt`) et de les afficher à la suite.

5°) Même question mais pour obtenir le nombre total de lignes.

Exercice 8 (Compression)
1°) Avec `tar` uniquement :
  a) Faites une archive compressée d'un répertoire de votre compte.
  b) Décompressez cette archive dans un répertoire nommé `decomp`.

2°) Même chose avec `tar` et `gzip`.

3°) Comparez les tailles des différents fichiers.

Exercice 9 (Filtres)

1°) Créez un fichier `ls.txt` qui contient le manuel de la commande `ls`.

2°) Transformez ce fichier en majuscule.

3°) Supprimez tous les caractères de ce fichier qui ne sont pas des voyelles, des espaces et des fins de lignes.

4°) En une seule commande et sans créer de fichier, écrivez le manuel de la commande `cd` sans espace et sans la lettre `e`.

Exercice 10 (Filtres complexes)

1°) Écrivez une commande qui affiche tous les processus autres que `bash` lancés par un utilisateur.

2°) Modifiez cette commande de façon à ce que cela soit maintenant tous les numéros de processus associés qui apparaissent.

3°) Écrivez une commande qui n'affiche que ce processus, puis que son numéro.

4°) Modifiez cette commande de façon à tuer tous les processus associés.

Exercice 11 (Composition de filtres)

1°) Écrivez une variante de `ls` qui n'affiche que les droits d'accès et les noms de fichiers.

2°) Écrivez une suite de filtres (par exemple, `cut`, `grep`, `ls`, ...) qui donne le nombre de liens symboliques dans un répertoire (par exemple `/bin`).

3°) Écrivez une commande qui affiche l'ensemble des utilisateurs et services qui font tourner des processus sur le système (`cut`, `ps`, `sort`, `uniq`).

4°) Écrivez un équivalent de la commande `ls -lS` à partir des commandes `ls -l` et `sort`.

Exercice 12 (Commande `awk`)

1°) Utilisez la commande `awk` pour afficher de cette façon le contenu du fichier `/etc/passwd` :

```
User : bin     id : 1
User : bin     id : 1
User : daemon  id : 2
...
```

2°) Écrivez une commande `awk` qui permet de n'afficher que les lignes d'un fichier dont le premier mot a entre 3 et 6 caractères.

Et maintenant ?

... `https://www.cours-gratuit.com/unix-linux`
(juste les commandes : pas les scripts) ...