Api/casoft Init - Jour 2 - Linux avancé

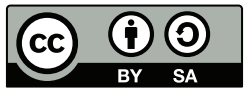
Devenir un pro de la ligne de commande

Stéphane Bonnet

Association Picasoft

Mardi 28 janvier 2020

picasoft



Sommaire

Pourquoi s'embêter à taper des commandes au lieu de cliquer sur des boutons ? Parce que la ligne de commande c'est...

Cool

Flexible Lève les limites des interfaces graphiques en permettant la *combinaison* de commandes

Rapide Plus économe en ressources et plus rapide à utiliser

Éducatif Permet de découvrir comment les choses marchent *vraiment*

Amusant Outil favori des *vrais* hackers !

Ligne de commande ? CLI ?

En anglais, on parle de *Command Line Interface* (CLI).

C'est aussi...

Puissant

- Permet d'automatiser efficacement et rapidement toutes les tâches
- Écriture des *scripts* : suites de commandes stockées dans un fichier. On pourra les ré-exectuter en lançant le script par le nom du fichier les ré-exécuter. Plus de détails demain !

Indispensable

- Permet de *diagnostiquer* et *corriger* les erreurs système
- Interface unique (en général) de commande pour les machines distantes...

Conclusion

La ligne de commande c'est cool, puissant et indispensable : c'est le couteau suisse de Linux !

L'interpréteur de commande

- L'interpréteur de commande. . . interprète les commandes
- C'est un programme comme un autre.
- Automatiquement lancé au démarrage d'un terminal ou après le *login* en mode console

```
ubuntu-18 login : bonnetst
Mot de passe :
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-43-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

bonnetst@ubuntu-18:~$
```

shell ?

On parle souvent de *shell* quand on fait référence à un interpréteur de commandes.

Shell

En toute rigueur, toute interface utilisateur pour les services d'un système d'exploitation est un shell.
Il peut être textuel ou graphique (cas des environnements graphiques de bureau comme Gnome).

On appelle *shells* les interpréteurs de commandes pour des raisons historiques... continuons !

Le prompt

Il indique que le shell attend une commande

```
$
```

Diffère selon le niveau de privilège : $ pour un utilisateur normal, # pour le super utilisateur *root*

```
# id -un
root
```

Il ne faut pas entrer le prompt. La touche “Entrée” termine la ligne et exécute la commande.

```
$ date
jeudi 17 janvier 2019, 10:06:58 (UTC+0100)
```

Structure de la ligne de commande

Le premier mot est la *commande* a exécuter

```
$ echo
```

Suivent un ou plusieurs arguments

```
$ echo foo bar
foo bar
```

Les commandes prennent souvent des options préfixées par un ou deux tirets.

```
$ date --utc
jeudi 17 janvier 2019, 09:54:51 (UTC+0000)
```

Espaces et casse

Les espaces ne comptent pas

```
$ echo foo       bar
foo bar
```

Sauf si on utilise les guillemets

```
$ echo "foo       bar"
foo       bar
```

La casse compte !

```
$ Echo foo
La commande « Echo » n'a pas été trouvée
...
$ echo foo
foo
```

Quitter le shell

Le shell est un programme, on peut le quitter.

Lancez un nouveau shell `bash` dans votre terminal Quittez-le au moyen de la commande `exit`.

Ça n'a pas fermé le terminal... Seul le nouveau shell a été quitté !

Quittez à nouveau le shell par `exit`. Le terminal se ferme.

`Ctrl+D`

`Ctrl+D` quitte également le shell en lui transmettant le caractère de contrôle `EOF` (*End Of File*). Essayez !

Commandes de base

Hier, on a vu quelques commandes de base dont :

- `ls` Liste les fichiers du répertoire courant
- `mv` Renomme ou déplace un fichier
- `mkdir` Crée un répertoire
- `rmdir` Supprime un répertoire
- `cp` Copie des fichiers
- `rm` Supprime un fichier

Il faut se souvenir au moins de celles-là !

Elles font partie du shell ?

La plupart des commandes sont des programmes indépendants lancés par le shell. Une commande directement fournie par le shell, comme `cd`, est dite *builtin*.

man

Une documentation complète est intégrée au système. On y accède par *man*.
`man` affiche le manuel complet d'une commande. Par exemple pour obtenir le manuel de `ls` :

```
$ man ls
```

Quelques pages utiles...

`man`	Le manuel du manuel
`intro`	Une introduction au système

man

Une fois dans `man`, quelques touches utiles :

Touche	Explications
q	Quitter l'aide
Espace	passe à la page suivante
/<motif>	Chercher <motif> dans la page
n	Chercher suivant
N	chercher précédent

Le déplacement se fait avec les touches fléchées.

A vous

Avec `man`, cherchez comment faire afficher à `ls` les fichiers du répertoire courant un par ligne, dans l'ordre de leur taille.

apropos

`apropos` permet de chercher par mot-clé :

```
$ apropos copy
[...]
cp (1)               - copy files and directories
cpgr (8)             - copy with locking the given file to the password or group file
cpio (1)             - copy files to and from archives
[...]
```

`man -k <mot>` assure la même fonction.

Directement dans les commandes

Les commandes contiennent souvent de l'aide, accessible en général par les options `-h` ou `--help`

```
$ cd --help
cd: cd [-L|[-P [-e]] [-@]] [rép]
    Change le répertoire de travail du shell.
[...]
```

Lisez la documentation !

Il est toujours utile de RTFM quand on est perdu...

Où sont les fichiers

Les fichiers sont contenus dans des répertoires qui forment une arborescence.

La racine de l'arborescence est repérée par un /
Ce répertoire est appelé répertoire racine, ou *root directory*

`cd` Change de répertoire
`pwd` Affiche le répertoire courant

Allons y

Allez dans / avec cd et affichez son contenu.

```
$ pwd
/home/bonnetst
$ cd /
$ pwd
/
$ ls
bin    dev   initrd.img      lib64       mnt   root  snap      sys  var
[...]
```

Chemins

L'emplacement d'un fichier ou un répertoire est repéré par son chemin dans l'arborescence.

Chemin absolu

Le chemin dans l'arborescence qui indique où se trouve un fichier **depuis la racine**

Chemin relatif

Le chemin dans l'arborescence qui indique où se trouve un fichier **depuis le répertoire courant**

Répertoires . et ..

Chaque répertoire contient un répertoire appelé '.' et un autre appelé '..'

.

Représente le répertoire courant

```
$ cd /usr/bin
$ pwd
/usr/bin
$ cd .
$ pwd
/usr/bin
```

..

Représente le répertoire parent

```
$ cd /usr/bin
$ pwd
/usr/bin
$ cd ..
$ pwd
/usr
```

Répertoire personnel

Chaque utilisateur dispose d'un répertoire personnel aussi appelé *home directory*.

Répertoire ~

Le répertoire ~ représente le home directory de l'utilisateur courant.

Remarque

La commande cd sans arguments revient au home directory de l'utilisateur.

```
$ cd /usr/bin
$ pwd
/usr/bin
$ cd ~
$ pwd
/home/bonnetst
```

```
$ cd /usr/bin
$ pwd
/usr/bin
$ cd
$ pwd
/home/bonnetst
```

Que contiennent les répertoires ?

L'arborescence est complexe mais organisée.

Chemin	Contenu
/	Racine
/bin, /sbin	programmes indispensables
/boot	noyau du système et bootloader
/dev	Fichiers spéciaux d'accès aux périphériques
/etc	Fichiers de configuration de la machine
/home	Répertoires personnels (en général)
/lib	Bibliothèques système
/media	Media amovibles (clés usb...)
/root	Répertoire personnel du superutilisateur
/tmp	Fichiers temporaires
/usr	Applications, programmes, etc.
/var	Spools, logs, caches etc.

Que contiennent les répertoires ?

Beaucoup d'autres répertoires...
Qui contiennent des sous-répertoires...

`tree`

```
cd /
tree -d
```

`tree` n'est pas installée par défaut. Installez-la avec `apt`.

C'est loooong

`CTRL+C` met fin à une commande (presque) sans lui demander son avis.
Utilisez-la quand vous en aurez assez de voir défiler des répertoires.

Pour aller plus loin

`man hier` documente tous les répertoires !

Liens symboliques : la commande `ln`

Certains fichiers ne sont que des pointeurs vers d'autres fichiers. Ce sont des *liens symboliques*.

Créez un fichier avec l'éditeur nano

```
cd
nano fichier.txt
```

Quittez nano avec `CTRL+X`

Créez un lien symbolique

```
ln -s fichier.txt lien.txt
```

```
$ ls -l
total 4
-rw-rw-r-- 1 bonnetst bonnetst  8 janv. 19 19:10 fichier.txt
lrwxrwxrwx 1 bonnetst bonnetst 11 janv. 19 19:12 lien.txt -> fichier.txt
$ cat fichier.txt
Salut!
$ cat lien_vers_fichier.txt
Salut!
```

Liens symboliques : la commande `ln`

Les liens ne contiennent que chemin vers la cible tel qu'il a été saisi !

```
$ mkdir subdir
$ ln -s fichier.txt subdir/lien.txt
$ ls -l subdir
total 0
lrwxrwxrwx 1 bonnetst bonnetst 11 janv. 19 19:28 lien.txt -> fichier.txt
```

Il y a du rouge, c'est mauvais signe !

```
$ ln -sf ../fichier.txt subdir/lien.txt
$ ls -l subdir
total 0
lrwxrwxrwx 1 bonnetst bonnetst 11 janv. 19 19:29 lien.txt -> ../fichier.txt
```

Quand on crée un lien, on utilise...

- Le chemin **absolu** vers la cible
- Le chemin relatif vers la cible **depuis l'emplacement du lien**

Commandes récursives

Traiter des sous-répertoires un par un est fastidieux. `mkdir`, `cp`, `rm` admettent des fonctionnement récursifs.

`mkdir -p`

Crée un répertoire et tous ses parents d'un coup

```
$ mkdir -p a/b
$ ls -d a/b
a/b
```

cp récursif

cp -R

Copie un répertoire et tous ses sous-répertoires d'un coup

```
$ cd a
$ cp /etc/fstab /etc/issue /etc/hosts .
$ ls
b  fstab  hosts  issue
$ cp -R . ~/c
$ ls ~/c
b  fstab  hosts  issue
```

cp -R crée automatiquement la destination si elle n'existe pas.

rm récursif

rm -R

Supprime un répertoire et tous ses sous-répertoires d'un coup

```
$ cd ~/c
$ ls
b fstab  hosts  issue
$ rm fstab hosts issue
$ cd ..
$ rmdir c
rmdir: impossible de supprimer 'c': Le dossier n'est pas vide
$ rm -R c
$ ls c$
ls: impossible d'accéder à 'c': Aucun fichier ou dossier de ce type
```

Danger, Will Robinson

Cette commande est très puissante. Il faut faire attention à ne pas supprimer des données par inadvertance.

Voir des fichiers

Inutile d'utiliser un éditeur pour afficher un fichier

cat

(conCATetnate) prend une liste de fichiers et "affiche" leur contenu.

```
$ cat /etc/issue
Ubuntu 18.04.1 LTS \n \l
```

less

Affiche un ou plusieurs fichiers. Utiliser les flèches pour faire défiler le texte, q pour quitter.

```
$ less /usr/share/dict/words
```

less et man

C'est `less` qui est utilisé par `man` pour afficher les pages de manuel.

Les fichiers standards

A l'exécution, chaque programme se voit attribuer trois canaux de communication :

- **stdin** représente tout ce qui est tapé au clavier
- **stdout** représente tout ce qui est affiché à l'écran
- **stderr** est une autre sortie destinée aux messages d'erreur

Descripteurs de fichier

Du point de vue du programme, ces canaux sont des fichiers particuliers. Un programme peut ouvrir de nombreux fichiers, identifiés par un descripteur de fichier.

Descripteur de fichier

Numéro associé à chaque fichier ouvert dans un programme. Utilisé par le programme pour accéder au fichier.

Descripteurs de fichiers et fichiers standard

Les fichiers standards sont associés dans tous les programmes aux trois premiers descripteurs.

Fichier	Descripteur
stdin	0
stdout	1
stderr	2

Redirections

Elles consistent à associer aux descripteurs standards d'autre fichiers que ceux par défaut.

Redirections de sortie Redirigent `stdout` vers un fichier

Redirections d'erreur Redirigent `stderr` vers un fichier

Redirections d'entrée Redirigent `stdin` depuis un fichier

Redirection de sortie : opérateurs > et >>

Opérateur >

Redirige `stdout` vers un fichier. Le contenu du fichier est préalablement effacé.

Opérateur »

Redirige `stdout` vers un fichier. Ajoute la sortie à la fin du fichier.

```
$ echo "Une première ligne" > fichier.txt
$ cat fichier.txt
Une première ligne
$ echo "Une seconde première ligne" > fichier.txt
$ cat fichier.txt
Une seconde première ligne
$echo "Une seconde ligne" >> fichier.txt
$ cat fichier.txt
Une seconde première ligne
Une seconde ligne
```

Redirection de sortie : cat

`cat` prend tout son sens avec les redirection de sortie. La commande

```
cat fichier1 fichier2 > fichier3
```

concatène fichier1 et fichier2 dans fichier3.

cat pour créer des fichiers

`cat` est très utile pour créer des fichiers quand on n'a pas d'éditeur : il suffit de rediriger sa sortie vers un nouveau fichier.

```
cat > newfile
```

Pour terminer, utiliser `CTRL+D`

Redirection d'entrée : opérateur <

Redirige le contenu d'un fichier vers `stdin`.

bc

`bc` est un programme de calcul en précision arbitraire. Utiliser `CTRL+D` ou "quit" pour quitter.

```
$ echo 1+1 > calcul
$ bc < calcul
2
```

`bc` a lu son entrée dans le fichier `calcul` et pas au clavier.

Redirection d'erreur

On peut indiquer optionnellement le descripteur redirigé.

Redirection de stderr

Il suffit d'indiquer dans la redirection que c'est le descripteur 2 qui est redirigé :

`ls toto 2> erreur`

```
$ ls foo
ls: impossible d'accéder à 'foo': Aucun fichier ou dossier de ce type
$ ls foo 2> erreurs
$ cat erreurs
ls: impossible d'accéder à 'foo': Aucun fichier ou dossier de ce type
```

Redirection vers un descripteur

On peut indiquer optionnellement le descripteur de destination

Redirection vers un descripteur

Il suffit d'indiquer le numéro de descripteur préfixé par `&`

Redirection de `stderr` vers `stdout`

`ls -lR / >filelist 2>&1`

Redirige la sortie standard dans filelist et la sortie d'erreur vers la sortie standard. Les erreurs de `ls` apparaîtront dans `filelist`.

Attention à l'ordre

`ls -lR / 2>&1 >filelist` ne produit pas le même résultat

Plus de détails sur les fichiers standard

L'autocomplétion

Accessible par `TAB` après avoir tapé le début de ce qu'on cherche.

Autocomplétion dans le système de fichiers

`TAB` complète les première lettres s'il n'y a pas d'ambiguïté

`TAB TAB` affiche une liste des possibilités d'autocomplétion

Autocomplétion en général

`TAB` peut aussi auto-compléter des commandes, des arguments, etc. Il faut en abuser !

Contexte

Sensible au contexte et ne présente que des propositions qui ont du sens... en général.

Éditer la ligne de commande

On peut se déplacer dans la ligne de commande avec les flèches et l'éditer. Inutile de tout recopier. `bash` permet aussi l'édition avec des raccourcis clavier.

Raccourci	Action
`ALT+f`	Avance d'un mot
`ALT+b`	Recule d'un mot
`CTRL+a`	Va en début de ligne
`CTRL+e`	Va en fin de ligne
`ALT+d`	Supprime jusqu'à la fin du mot
`ALT+Backspace`	Supprime jusqu'au début du mot
`CTRL+y`	Réinsère la suppression
`CTRL+k`	Supprime la fin de ligne
`CTRL+u`	Supprime de début de ligne
`CTRL+_`	Annule le dernier changement

RTFM

Il y a beaucoup d'autres raccourcis... lisez la doc : `man bash`.

Rappeler des commandes

`bash` maintient un historique des commandes et permet de les rappeler pour ne pas avoir à les retaper. L'historique est stocké dans le fichier `~/.bash_history`.

- La commande "`history`" affiche l'historique.
- Les touches haut et bas permettent de naviguer dans l'historique.
- Le point d'exclamation suivi d'une chaîne rappelle la dernière commande commençant par cette chaîne.
- La combinaison `CTRL+R` entre en mode recherche dans l'historique. `CTRL+R` recherche à nouveau.

Astuce

La commande `!!` relance la dernière commande. On peut l'utiliser avec `sudo` pour relancer une commande en superutilisateur :

```
sudo !!
```

Faire référence à plusieurs fichiers

C'est fastidieux d'être obligé d'écrire le nom de tous les fichiers dans les commandes. On peut se simplifier la vie grâce aux globbing patterns.

globbing pattern (*motif d'englobement*)

Un motif de nom de fichier qui en englobe plusieurs.

wildcard (*joker*)

Lettre spéciale qui représente un caractère ou une suite de caractères quelconques dans un *globbing pattern*.

Le joker “*”

Joker le plus utilisé. Remplace toute suite de caractères dans un motif. S'il est utilisé seul, remplace tout mot.

Créons des fichiers

```
cd ; mkdir api ; cd api
touch here that there this
```

Le point-virgule

Le point virgule sépare des commandes successives sur la ligne de commande.

`touch`

Crée les fichiers passés en arguments. Si un fichier existe déjà, met à jour sa date de dernière modification.

Le joker “*”

Supprimons des fichiers

On pourrait écrire : `rm here that there this`
`rm *` fait la même chose car le joker “*” correspond à tous les noms de fichiers.

```
$ ls
here that there this
$ rm *
$ ls
$
```

Le joker “*”

Re-créeons des fichiers

```
touch here that there this
```

On peut restreindre le motif en donnant une partie du nom.

```
$ ls *
here that there this
$ ls t*
that there this
$ ls *h*
here that there this
$ ls th*e
there
$ ls that*
that
$ ls thi*s
this
```

Attention

“*” correspond aussi aux séquences vides !

Le joker " ?"

Il représente un caractère unique quelconque dans un motif.

```
$ ls th?t
that
$ ls th??
that this
$ ls th???
there
$ ls ?here
there
$ ls *here        <---- Attention ici le joker correspond aussi à une séquence vide
here there
```

Le joker "[]"

Permet de définir un ensemble de caractères pouvant apparaître dans le nom.

Créons plus de fichiers

```
touch file_1 file_2 file_3 file_4 file_a file_b file_c
file_d
```

```
$ ls file_[12ab]
file_1 file_2 file_a file_b
```

Le joker "[]"

On peut spécifier des intervalles plutôt que de donner tous les caractères.

```
$ ls file_[1-3b-d]
file_1 file_2 file_3 file_b file_c file_d
```

On peut aussi prendre le complément de la liste en la préfixant par un "^".

```
$ ls file_[^1-3b-d]
file_4 file_a
```

Échappement de caractères

Et si il y a un fichier dont le nom contient le caractère "-" ?

Un fichier de plus

`touch file_-`

```
$ ls file_[1-3b-d]
file_1 file_2 file_3 file_b file_c file_d
$ ls file_[1-3b-d\-]
file_- file_1 file_2 file_3 file_b file_c file_d
```

Caractère d'échappement

Le caractère "\" supprime l'interprétation par le shell du caractère qui le suit. On peut même échapper l'espace.

```
$ touch my\ file
$ ls
'my file'           <----- Les simple quotes indiquent qu'il s'agit d'un seul fichier
```

Plus de formes d'expansion...

Remarques

Jokers dans les chemins

Le chemin fait partie du nom des fichiers, on peut mettre des jokers dans ceux-ci.

`ls /var/*/*.log`

liste tous les fichiers de logs dans tous les sous-répertoires directs de `/var`

`ls /var/*/*/*.log`

liste tous les fichiers de logs dans tous les sous-répertoires des sous-répertoires de `/var`

Si le motif ne correspond à aucun fichier ?

Dans ce cas, le motif est transmis tel quel à la commande.

```
$ rm file?
rm: impossible de supprimer 'file?': Aucun fichier ou dossier de ce type
```

Remarques

Attention aux noms pris comme des options

On suppose qu'il existe un fichier "-R".
Si on exécute "`rm *`", l'englobement va passer le nom `-R` à "`rm`" qui l'interprétera comme une option.
La commande réellement exécutée sera "`rm -R *`". Oups !

Séparer les options des arguments

- "`--`" est utilisé dans les commandes pour séparer explicitement les options des arguments.
- Tout ce qui suit "`--`" est interprété comme un argument.

"`rm -- *`" est donc plus sûr, puisque c'est bien la commande "`rm -- -R *`" qui sera exécutée.

Et les fichiers cachés ?

Des fichiers cachés ? Où ça ?

Tous les fichiers dont le nom commence par un point sont cachés.

- Ils ne sont pas affichés par "`ls`".
- Ils ne sont pas considérés pour l'englobement.

```
$ touch .hidden
$ ls
$ ls -a
.  ..  .hidden
$ rm *
$ ls -a
.  ..  .hidden
```

Et les fichiers cachés ?

shopt

`shopt` permet de changer certaines options du shell.

- `shopt -s <option>` active l'option `<option>`
- `shopt -u <option>` désactive l'option `<option>`

L'option `dotglob`

Si elle est active, les fichiers commençant par un point sont candidats à l'englobement. Par défaut désactivée.

```
$ ls -a
.  ..  .hidden
$ shopt -s dotglob
$ rm *
$ ls -a
.  ..
```

Trouver des fichiers

Des centaines de milliers de fichiers sur une distribution classique.
Difficile de s'y retrouver sans outils...

`locate`

Recherche des fichiers dans une base de données préremplie

`find`

Recherche des fichiers dans l'arborescence, éventuellement avec des filtres précis.

locate

`locate <motif>` recherche `<motif>` dans la base de données `/var/lib/mlocate/mlocate.db`

```
$ locate issue
/etc/issue
/etc/issue.net
[...]
```

Si `locate` ne renvoie rien

- Rien n'a été trouvé
- La base de donnée est vide

updatedb

`updatedb` met à jour la base de données `/var/lib/mlocate/mlocate.db`

sudo obligatoire !

Cette commande modifie un fichier que seul le superutilisateur peut modifier.
Il faut exécuter `sudo updatedb`.

sudo (*SuperUserDO*)

- Cette commande exécute une autre commande avec les droits du superutilisateur.
- Seuls certains utilisateurs ont le droit de l'invoquer.
- Elle demande le mot de passe de l'utilisateur qui l'invoque avant de s'exécuter.

find

`find` recherche un fichier dans le système de fichiers.

Il suffit de spécifier le point de départ et ce qu'on cherche.

```
$ find . -name file_1
./api/file_1
```

On peut utiliser des jokers dans le nom

```
$ find . -name 'file_1*' -print
./api/file_11
./api/file_18
./api/file_1
[...]
```

Attention aux quotes

Le motif doit être entre quotes, sinon il sera interprété par le shell.

Filtrer avec `find`

`find` est capable de filtrer la recherche

- Par type de fichier
- Par taille
- Par date de modification
- ...

Cherchons les fichiers de plus de 10 Mo

```
find / -size +10M -print
```

Il y a plein d'erreurs !

Elles sont dûes au permissions d'accès à certains fichiers. On peut les supprimer en redirigeant la sortie d'erreur vers le périphérique 'poubelle' `/dev/null` :

```
find / -size +10M -print 2> /dev/null
```

Exécuter des commandes avec `find`

`find` peut exécuter une commande sur chaque fichier trouvé avec l'option `-exec`.

Créer un backup de chaque fichier

```
find api -name 'file*' -exec cp {} {}.bak \;
```

Explication

- `cp` est exécutée pour chaque fichier trouvé
- `{}` est remplacé par le nom du fichier trouvé
- `\;` indique la fin de la commande.

- `-exec` n'autorise qu'une seule commande.
- On peut avoir plusieurs `-exec`.

grep

`grep` recherche un motif dans des fichiers

Chercher tous les mots qui contiennent "aard"

```
grep 'aard' /usr/share/dict/words
```

```
$ grep 'aard' /usr/share/dict/words
Kierkegaard
Kierkegaard's
aardvark
aardvark's
aardvarks
```

Remarques

- `grep` peut être utilisée récursivement : option `-R`
- `grep` sans argument, cherche le motif sur l'entrée standard

Les motifs sont en fait des expressions régulières...

file

`file` identifie la nature des données d'un fichier.

```
$ file /usr/lib/libreoffice/program/intro.png
/usr/lib/libreoffice/program/intro.png: PNG image data, 661 x 169, 8-bit/color RGBA, interlaced

$ file /bin/bash
/bin/bash: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0,
BuildID[sha1]=b636f50d85c3cca7cf2518030446660c1d90d660, stripped
```

Remarque

`file` fonctionne en analysant réellement le contenu du fichier, il ne se sert pas de l'extension.

Un peu de manipulation de fichiers

Il existe de nombreuses commandes de manipulation de fichiers utiles :

- `wc` Compte les lignes, mots, etc. d'un fichier
- `sort` Trie les lignes d'un fichier
- `uniq` Supprime les doublons adjacents d'un fichier
- `tr` Transforme les caractères d'un fichier
- `diff` Compare deux fichiers et affiche les différences
- `cut` Découpe un fichier en champs
- `sed` Édite un fichier à la volée

tr

tr Transforme des caractères entre deux ensembles

```
$ cd ~/api ; echo 'The quick brown fox jumps over the lazy dog' > file.txt
```

Un peu de cryptographie...

```
$ tr 'a-z' 'b-za' < file.txt
Tif rvjdl cspxo gpy kvnqt pwfs uif mbaz eph
```

Passage en capitales...

```
$ tr '[:lower:]' '[:upper:]' < file.txt
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
```

Séparons les mots...

```
$ tr '[:blank:]' '\n' < file.txt
The
quick
brown
fox
jumps
over
the
lazy
dog
```

Séquences d'échappement

Interprétées par le shell comme des caractères spéciaux : \n est un retour à la ligne, \t une tabulation...

cut

`cut` Supprime des sections de chaque ligne d'un fichier.

On garde les caractères 5 à 9...

```
$ cut -c 5-9 file.txt
quick
```

On garde les champs 3 et 5...

```
$ cut -f 3,5 -d' ' file.txt
brown jumps
```

Délimiteur

Le délimiteur de champs est par défaut la tabulation. L'option `-d` permet de le changer.

sed

`sed` (*Stream EDitor*) Édite le fichier ligne par ligne à la volée.

Remplaçons *brown* par *yellow*...

```
$ sed 's/brown/yellow/g' file.txt
The quick yellow fox jumps over the lazy dog
```

- `s` est la commande de remplacement. Elle prend deux paramètres :
  - le motif à rechercher (c'est une regex)
  - le texte à substituer aux correspondances
- `g` indique que le remplacement affecte toutes les occurrences.
- Le séparateur est le caractère qui suit directement la commande `s`. On aurait pu utiliser un autre caractère que "/".

RTFM

`sed` est une commande très puissante mais très complexe. Lisez le man.

Les Pipes

Généralités

KISS (Keep It Simple, Stupid)

Les commandes, aussi complexes qu'elles soient, sont chacune spécialisées dans une fonction spécifique. C'est la philosophie générale des commandes Unix : ne faire qu'une chose, mais la faire bien.

Les pipes (ou tubes)

- Connectent la sortie standard d'une commande à l'entrée standard d'une autre.
- Permettent de chaîner des commandes pour construire des traitements complexes.

Les Pipes

Comment ça marche ?

Supposons qu'on veuille voir le listing complet des fichiers du système. On peut exécuter "`ls -lR /`" mais la sortie est trop grande et on ne pourra pas tout voir...
On peut utiliser les services de `less` pour paginer la sortie.
Il suffit de taper :

```
ls -lR / | less
```

Le symbole "|" représente le pipe entre les deux commandes.

Les Pipes

Remarques

- Les pipes ne sont pas de taille infinie. Les commandes qui produisent les données (ici `ls`) attendent lorsqu'ils sont pleins. **On ne peut pas perdre de données dans un pipe.**
- Si on arrête le consommateur (ici `less`), le producteur se termine également par fermeture de son fichier de sortie standard.
- De même, la fin du producteur entraine la fermeture de l'entrée standard du consommateur et donc son arrêt.

Les Pipes

La commande tee

On peut vouloir enregistrer la sortie d'une commande dans un fichier en même temps qu'on la transmet à une autre commande par un pipe.

tee

`tee` lit son entrée standard, l'enregistre dans un fichier et la recopie sur sa sortie standard.

Enregistrer la sortie de `ls -lR` pendant qu'on compte les fichiers

```
ls -lR ~ | tee listing.txt | wc -l
```

Les Pipes

À vous de jouer !

- Téléchargez “Vingt mille lieues sous les mers” de Jules VERNE depuis le site du projet Gutenberg :

```
wget http://www.gutenberg.org/files/54873/54873-0.txt
```

Combien de mots différents sont utilisés par l'auteur ?

Il faut : lire le fichier, supprimer la ponctuation et les caractères spéciaux (`,.;!-()\r_«»?-"`),le transformer pour qu'il ait un mot par ligne, tout passer en minuscules, trier les mots, éliminer les mots en double ou plus, compter le nombre de lignes.

Combien de fois le Capitaine Nemo est-il nommé ?

L'approche est similaire, mais il faut aussi ne garder que le mot “Nemo”.

Les processus

Définition

Qu'est-ce que c'est ?

- Un processus est un programme en cours d'exécution.
- Un programme est un exécutable binaire stocké sur le disque dans un fichier
- A son exécution, le programme est copié en mémoire par le système et devient un processus.
- De nombreux processus s'exécutent en même temps : c'est une des fonctions principales du système de partager le processeur et la mémoire entre ceux-ci.

Les processus

Hierarchie des processus

PID

Les processus sont identifiés par in nombre : l'identifiant du processus, ou *Process ID (PID)*.

Arborescence des processus

Un processus a un père unique : c'est le processus qui l'a créé. L'ID du processus père est le *Parent Process ID (PPID)*.

Processus initial

Le premier processus est exécuté directement par le noyau au démarrage. Il s'agit du processus `init` (`systemd` sur Ubuntu). Son PID est 0 et il est l'ancêtre de tous les processus.

Les processus

État des processus

A tout instant, un processus est dans l'un des états suivants :

- R (Running) : Processus éligible au processeur
- S (Sleeping) : Processus en attente d'un événement
- D (Uninterruptible Sleep) : Processus en attente d'une entrée / sortie
- T (Stopped) : Processus interrompu par un signal de contrôle
- Z (Zombie) : Terminé, en attente du processus père

Les processus

Voir les processus

ps

`ps` affiche les processus en cours d'exécution et les informations associées

`ps` seul affiche tous les processus de l'utilisateur associé au terminal :

```
$ ps
  PID TTY          TIME CMD
10923 pts/2    00:00:01 bash
12472 pts/2    00:00:00 cat
12622 pts/2    00:00:00 ps
```

Il y a ici trois processus. Les champs s'interprètent comme suit :

- **PID** Le PID
- **TTY** Le terminal de contrôle
- **TIME** Le temps réellement passé à utiliser le processeur
- **CMD** La commande exécutée

Les processus

Voir les processus

La commande "ps a" affiche tous les processus de l'utilisateur.

```
$ ps a
  PID TTY      STAT   TIME COMMAND
  986 tty1     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_S
  988 tty1     Sl+   21:46 /usr/lib/xorg/Xorg vt1 -displayfd 3 -auth /run/user/
  999 tty1     Sl+    0:00 /usr/lib/gnome-session/gnome-session-binary --sessio
 1163 tty1     Sl+   41:15 /usr/bin/gnome-shell
 1208 tty1     Sl    14:18 ibus-daemon --xim --panel disable
 1212 tty1     Sl     0:00 /usr/lib/ibus/ibus-dconf
 [...]
```

La commande "ps aux" affiche tous les processus de tous les utilisateurs.

Les processus

Voir les processus

pstree

La commande `pstree` affiche l'arbre des processus en cours d'exécution.

```
$ pstree -A
systemd-+-ModemManager---2*[ModemManager]
        |-NetworkManager-+-dhclient
        |                `-2*[NetworkManager]
        |-2*[VBoxClient---VBoxClient]
        |-2*[VBoxClient---VBoxClient---VBoxClient]
        |-VBoxClient---VBoxClient---2*[VBoxClient]
        |-VBoxService---7*[VBoxService]
        |-accounts-daemon---2*[accounts-daemon]
        |-acpid
        |-avahi-daemon---avahi-daemon
        |-boltd---2*[boltd]
        |-colord---2*[colord]
        |-cron
[...]
```

`systemd` est bien l'ancêtre de tout les processus...

Les processus

Suivre les processus en temps réel

`top`, `htop`

- `top` affiche l'évolution des processus en temps réel. Quitter avec "q".
- `htop` fait la même chose avec une interface plus moderne.

Ces deux commandes montrent aussi en temps réel l'utilisation des processeurs et l'occupation de la mémoire. Elles permettent de suivre les performances du système.

Les processus

Les signaux

Signal

Le noyau peut transmettre des signaux aux processus pour des raisons diverses. Les processus peuvent intercepter certains de ces signaux. S'il ne le font pas, alors les signaux ont des effets par défaut sur le processus.

Transmission de signaux par le clavier

Quand un processus est exécuté au premier plan, certaines combinaisons de touches provoquent la transmission de signaux vers celui-ci.

- `CTRL+C` envoie SIGINT. Met fin au processus avec potentielle perte de données.
- `CTRL+Z` envoie SIGTSTP. Stoppe le processus et rend la main au shell.
- `CTRL+]` envoie SIGQUIT. Met fin au processus plus brutalement que `CTRL+C`.

Les processus

Effet des signaux

CTRL+C vs. CTRL+]

- Installez `sl` : `sudo apt install sl`
- Lancer sl. Le train passe.
- Observez la différence entre CTRL+C et CTRL+] pendant le passage du train.

Exécution en arrière plan

Une commande exécutée en arrière plan est dite *détachée* du terminal.

Exécution en arrière plan

Faire suivre la commande de "&". La commande se lance et rend la main au shell.
"`gedit &`" lance l'éditeur de texte gedit et rend la main.

Arrière plan et signaux

`CTRL+Z` suspend le processus attaché au terminal.
"bg" le relance en arrière plan, comme si on l'avait exécuté avec "&".

```
$ xeyes
^Z
[1]+  Arrêté                 xeyes
$ bg
[1]+ xeyes &
$
```

Mettre fin à des processus

kill

Pour mettre fin à un processus, la commande à utiliser est `kill` suivie du PID du processus à tuer.

```
$ ps
  PID TTY          TIME CMD
13154 pts/5    00:00:00 bash
13162 pts/5    00:00:01 xeyes
13178 pts/5    00:00:00 ps
$ kill 13162
$
[1]+  Complété                 xeyes
$
```

Processus récalcitrants

Par défaut, `kill` envoie `SIGTERM` au processus. Si le processus ne se termine pas avec ce signal,
"`kill -KILL`" (ou "`kill -9`") suivie du PID y mettra fin systématiquement mais avec plus de risques de perte de données.

Gestion des jobs

- `jobs` affiche la liste des processus en arrière plan
- `fg` ramène le processus indiquer en avant plan
- `bg` passe le dernier processus arrêté en arrière plan
- `kill` peut aussi s'utiliser avec un numéro de job.

```
$ xeyes &
[1] 13507
$ jobs
[1]+  En cours d'exécution    xeyes &
$ kill %1
$
[1]+  Complété                xeyes
$
```

Utilisateurs et groupes

Vocabulaire

Unix est un système multi-utilisateurs. Les utilisateurs peuvent appartenir à des groupes.

Utilisateur

Deux catégories :

- Utilisateurs *système* utilisés pour la sécurité du système.
- Utilisateurs *normaux* qui sont voués à se servir du système.

Un utilisateur est identifié par un nombre, l'**uid** (*user id*) auquel est associé un nom, le **username** ou **login**.

Groupe

Les utilisateurs appartiennent à un groupe principal et peuvent appartenir à plusieurs groupes secondaire. Les groupes sont définis par un numéro, le **gid** (*group id*) auquel est associé un nom.

Utilisateurs et groupes

Fichiers associés

Les fichiers de gestion des utilisateurs sont stockés dans le répertoire `/etc`.

- Le fichier `/etc/passwd` contient la liste des utilisateurs.
- Le fichier `/etc/shadow` contient les mots de passe chiffrés des utilisateurs.
- Le fichier `/etc/groups` contient la liste des groupes et de leurs membres.

Utilisateurs et groupes

Utilisateurs et groupes propriétaires

Les fichiers et répertoires sont tous la propriété d'utilisateurs variables.
L'immense majorité des fichiers appartient à l'utilisateur *root*.
La commande "`ls -l`" permet d'afficher le propriétaire d'un fichier.

```
$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1296 janv.  4 12:15 /etc/shadow
```

Ici, le fichier `/etc/shadow` appartient à l'utilisateur `root` et au groupe `shadow`.

Utilisateurs et groupes

Changement de propriétaire

Seul le superutilisateur peut changer le propriétaire de fichiers ou de répertoires au moyen des commandes suivantes :

- `chown` change l'utilisateur propriétaire (et éventuellement le groupe)
- `chgrp` change le groupe propriétaire

```
chown newuser:newgroup /my/file chgrp newgroup /my/file
```

Qui suis-je ?

Les commandes suivantes permettent d'identifier l'utilisateur courant et les groupes auxquel il est associé.

`id`

`whoami`

`groups`

```
$ id
uid=1000(bonnetst) gid=1000(bonnetst) groupes=1000(bonnetst),
4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),116(lpadmin),126(sambashare)
$ whoami
bonnetst
$ groups
bonnetst adm cdrom sudo dip plugdev lpadmin sambashare
```

Devenir grand chef

Utilisateurs et processus

- Un nouveau processus hérite de l'identité du processus qui l'a lancé.
- S'il est lancé par le shell d'un utilisateur, il hérite de l'identité de cet utilisateur.
- Le processus ne peut accéder qu'aux ressources qui sont accessibles à cet utilisateur.

Superutilisateur : `root`

Un utilisateur a accès à toutes les ressources du système : l'utilisateur `root` d'uid 0.

Devenir grand chef

sudo et su

Sur Ubuntu, `root` ne peut pas se connecter directement, il faut un moyen pour qu'un utilisateur normal puisse accéder aux ressources et fichiers protégés.

`sudo`

- Exécute une commande sous l'identité du superutilisateur.
- Ne peut être utilisée que par les membres du groupe `sudo`.
- "`sudo bash`" lance un shell superutilisateur.

`su`

- Sur d'autres systèmes, la commande `su` (*Substitute User*) est utilisée.
- Substitue l'utilisateur `root` à l'utilisateur courant.
- Nécessite d'entrer le mot de passe de l'utilisateur root.

Création et suppression

- `adduser` Ajoute un utilisateurs
- `deluser` Supprime un utilisateur
- `addgroup` Ajoute un groupe
- `delgroup` Supprime un utilisateur

Ajouter et supprimer des utilisateurs aux groupes

- `adduser user group` ajoute l'utilisateur `user` au groupe `group`
- `deluser user group` supprime l'utilisateur `user` du groupe `group`

Changer son mot de passe

La commande “`passwd`” permet de changer son mot de passe.

Changement pour autrui

Le superutilisateur peut changer le mot de passe d'autres utilisateurs.
“`passwd user`” change le mot de passe de l'utilisateur `user`.

Les permissions

Les fichiers et répertoires appartiennent à un utilisateur et à un groupe. Les permissions d'accès sont définies pour :

- Le propriétaire du fichier (*user*) : **(u)**.
- Le groupe du fichier : **(g)**.
- Le reste du monde (*others*) : **(o)**.

Les permissions d'accès peuvent être en :

- Lecture (*read*) : **(r)**.
- Écriture (*write*) : **(w)**.
- Exécution (*eXecute*) : **(x)**.

Visualisation des permissions

"`ls -l`" affiche les permissions d'un fichier ou d'un répertoire.

```
$ cd ~/api
$ ls -l
-rw-r--r-- 1 bonnetst bonnetst    0 janv. 27 21:03 file_1
drwxr-xr-x 2 bonnetst bonnetst 4096 janv. 27 21:03 repertoire
```

Un répertoire est repéré par le `d` en début de permissions.

Interprétation

- -rw-r--r-- indique les permissions pour le propriétaire
- -rw-r--r-- indique les permissions pour le groupe
- -rw-r--r-- indique les permissions pour les autres
- Dans le cas des **fichiers**, elles indiquent ce qu'on a le droit de faire *dans le fichier*
- Dans le cas des **répertoires**, elles indiquent ce qu'on a le droit de faire *dans le répertoire*
  - `r` permet de lister le contenu du répertoire
  - `w` permet de créer ou de supprimer des fichiers (indépendamment des permissions du fichier !)
  - `x` permet de se placer dans le répertoire ou de le traverser sur un chemin.

chmod

`chmod` affecte (=), ajoute (+) ou supprime (-) des permissions

```
$ cd ~/api
$ ls -l
-rw-r--r-- 1 bonnetst bonnetst    0 janv. 27 21:03 file_1
$ chmod g+w file_1
$ ls -l
-rw-rw-r-- 1 bonnetst bonnetst    0 janv. 27 21:03 file_1
$ chmod g=r file_1
$ ls -l
-rw-r--r-- 1 bonnetst bonnetst    0 janv. 27 21:03 file_1
$ chmod g-r file_1
$ ls -l
-rw----r-- 1 bonnetst bonnetst    0 janv. 27 21:03 file_1
```

On peut les combiner :

```
$ cd ~/api
$ chmod u+x,g+wx,o-r file_1
$ ls -l
-rwxrwx--- 1 bonnetst bonnetst    0 janv. 27 21:03 file_1
```

Les variables

Le shell permet de définir et d'utiliser des variables.

Variable

- Identifiée par son nom
- Affectée d'une valeur
- Elle peut être locale au shell
- Elle peut être exportée. Dans ce cas, on parle de variable d'environnement.

Manipulation des variables

- `myvar="hello"` Crée la variable `myvar` et y affecte la chaîne "hello".
- `export MYVAR="hello"` Crée la variable d'environnement `MYVAR`.
- `unset myvar` Supprime la variable `myvar`.
- `$myvar` accède à la valeur de la variable `myvar`.

Les variables

Exemple

```
$ myvar="hello"
$ echo $myvar
hello
```

Export de variables

Export

Une variable exportée est connue de tous les processus lancés à partir du shell où elle a été définie. Elle est héritée par tous les processus enfants. Elle fait partie de leur environnement.

```
$ myvar="hello"
$ export MYVAR="hello export"
$ bash   <----- A partir de ce point, on est dans un second shell
$ echo $myvar

$ echo $MYVAR
hello export
```

Variable PATH

- Spécifie les chemins où seront recherchées les commandes
- Variable extrêmement importante. Si un répertoire qui contient des commandes n'est pas dans le PATH, il faudra spécifier son chemin complet pour pouvoir la lancer.

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
$ which ls
/bin/ls
```

`which`

`which` affiche l'emplacement d'une commande. Recherche dans tous les répertoires du PATH.

Variable PS1

- Spécifie le format du prompt du shell.
- On peut tenter de la redéfinir pour personnaliser sont shell.
- RTFM !

Valeurs de retour

Valeur de retour

- Toutes les commandes retournent une valeur qui indique qu'elles ont réussi ou échoué.
- La valeur 0 indique le succès, toutes les autres valeurs l'échec.
- La variable du shell $? contient cette valeur à la fin de chaque commande.

```
$ true
$ echo $?
0
$ false
$ echo $?
1
```

Les différents fichiers de configuration de bash

bash exploite plusieurs fichiers de configuration :

- /etc/bash.bashrc est le fichier de configuration global, commun à tous les utilisateurs
- ~/.bashrc est le fichier de configuration spécifique à l'utilisateur

.bashrc

Les commandes de ce fichier sont exécutées par bash au démarrage.
Il suffit donc d'y ajouter toute personnalisation pour qu'elle soit prise en compte.

6 Plus de détails...

- Plus sur les descripteurs...
- Plus sur les expansions
- Expressions régulières

Les fichiers stdin, stdout, stderr existent-ils ?

Sous Unix, tout est un fichier : données, programmes, périphériques comme le clavier et l'écran sont des fichiers.

`tty`

La commande `tty` permet de voir à quel fichier sont associés les entrées et sorties standard.
Il s'agit du terminal de contrôle du shell (*controlling tty*)

```
$ tty
/dev/pts/0
```

Les fichiers de périphérique

Les fichiers de périphériques (*device files*) comme `/dev/pts/0` sont des interfaces avec le noyau Linux.
Y accéder déclenche l'exécution d'un service du noyau, comme lire des caractères depuis le clavier ou afficher des caractères à l'écran, au travers d'un sous-système du noyau ou d'un pilote de périphérique.

`/dev`

Le répertoire `/dev` contient tous les fichiers de périphériques. Ils peuvent être créés à la volée, par exemple après connexion d'une clé USB.

Jouons avec /dev/pts

Ouvrez deux terminaux. Dans chaque, lancez `tty` pour observer leur terminal de contrôle.

Écrivez dans le terminal en redirigeant vers le terminal de contrôle de **l'autre** terminal

```
$ tty
/dev/pts/0
$ echo Salut > /dev/pts/1
```

Le texte tapé dans l'autre terminal s'affiche ici !

```
$ tty
/dev/pts/1
$ Salut
```

Retour aux redirections

Expansion d'accolades

Expansion d'accolades

bash supporte l'expansion d'accolades : générer toutes les chaînes dans un intervalle ou à partir d'une liste.

```
$ echo {1..20}
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
$ echo a{c,d,e}f
acf adf aef
```

Cette expansion peut être utilisée dans des motifs.

```
$ touch file_{1..20}
$ ls
file_1   file_12  file_15  file_18  file_20  file_5  file_8
file_10  file_13  file_16  file_19  file_3   file_6  file_9
file_11  file_14  file_17  file_2   file_4   file_7
$ rm file_{12..14}
$ ls
file_1   file_15  file_18  file_20  file_5  file_8
file_10  file_16  file_19  file_3   file_6  file_9
file_11  file_17  file_2   file_4   file_7
```

Expansion d'accolades

Extensions multiples

L'expansion d'accolade est très utilisée pour sélectionner des fichiers avec des extensions multiples.
`ls *.{jpg,png,bmp}`
listera tous les fichiers correspondants à ces extensions.

Extensions

- Les extensions n'ont pas de sens particulier sous UNIX.
- Spécificité de Windows et de ses prédécesseurs.
- Pratique pour indiquer le type des fichiers...

Retour aux expansions

Expressions régulières (*regex*)

- Chaîne de caractères qui décrit précisément un ensemble de chaînes de caractères possibles.
- Dans les commandes qui utilisent un motif (comme `grep`), le motif est presque toujours une expression régulière.

Expressions régulières (*regex*)

Bases

- Décrit la suite de caractères qui doivent correspondre dans une chaîne.
- Les alternatives sont séparées par une barre verticale : `a|b` reconnaît tout mot contenant 'a' ou 'b'.
- Les parenthèses regroupent les expressions :
  - `ab|cd` reconnaît tout mot contenant 'ab' ou 'cd'
  - `a(b|c)d` reconnaît tout mot contenant 'abd' ou 'acd'.

Attention aux caractères interprétés par le shell

Les parenthèses et les barres verticales sont interprétées par le shell. Il faut les échapper pour les utiliser dans des regex. Pour exploiter le motif `a(b|c)d` avec grep, il faudra écrire :

```
grep 'a\(b\|c\)d'
```

Expressions régulières (*regex*)

Caractères

Chaque caractère de la chaîne à reconnaître doit être spécifié, soit en donnant sa valeur directement ou en donnant sa classe.

- `abcd` : la suite 'abcd'.
- `[ab]bcd` : soit a, soit b suivi de bcd. Les crochets définissent une classe de caractères acceptables pour un caractère donné.
- `[a-z]bcd` : on peut utiliser des intervalles dans les classes.
- `a.b` : trois caractères commençant par a et finissant par b. Le point correspond à n'importe quel caractère.
- `[:alnum:]` : tout caractère alphanumérique
- `[:digit:]` : tout chiffre
- `[:space:]` : tout espace

Expressions régulières (*regex*)

Quantificateurs et prédicats

Il représentent le nombre de répétitions d'une expression et ajoutent des contraintes sur sa position dans la chaîne.

- `expr*` : le motif `expr` répété 0 ou plusieurs fois.
- `expr+` : le motif `expr` répété 1 ou plusieurs fois.
- `expr?` : le motif `expr` répété 0 ou une fois.
- `^expr` : le motif `expr` en début de ligne.
- `expr$` : le motif `expr` en fin de ligne.

grep et expressions régulières

Trouver tous les mots commençant par ab

```
grep '^ab' /usr/share/dict/words
```

Trouver tous les mots de quatre lettres commençant par g et finissant par t

```
grep '^g..t$' /usr/share/dict/words
```

Trouver tous les mots commençant par g et finissant par t

```
grep '^g.\+t$' /usr/share/dict/words
```

C'est très pratique...

...pour les mots croisés !

Retour à grep