Linux_OpenClassRoom Documentation

Eric

nov. 04, 2018

Table des matières

CHAPITRE 1

Raccourcis clavier divers

Ctrl + R search for previous command

Ctrl + L efface le contenu de la console. Utile pour faire un peu de ménage quand votre console est encombrée, ou quand votre boss passe derrière vous et que vous n'aimeriez pas qu'il voie ce que vous étiez en train de faire. À noter qu'il existe aussi une commande, clear, qui fait exactement la même chose.

Ctrl + D envoie le message EOF (fin de fichier) à la console. Si vous tapez ce raccourci dans une ligne de commande vide (c'est-à-dire sans avoir écrit un début de commande au préalable), cela fermera la console en cours. À noter qu'il existe aussi la commande exit qui a le même effet.

Shift + PgUp vous permet de « remonter » dans les messages envoyés par la console. En mode graphique, la molette de la souris accomplit aussi très bien cette action. La touche Page Up est généralement représentée sur votre clavier par une flèche directionnelle Haut barrée de plusieurs petites lignes horizontales.

Shift + PgDown pareil, mais pour redescendre.

Ctrl + A ramène le curseur au début de la commande. La touche Origine a le même effet (elle est située à côté de la touche Fin et représentée par une flèche pointant en haut à gauche).

Ctrl + E ramène le curseur à la fin de la ligne de commandes. La touche Fin a le même effet.

Ctrl + U supprime tout ce qui se trouve à gauche du curseur. Si celui-ci est situé à la fin de la ligne, cette dernière sera donc supprimée.

Ctrl + K supprime tout ce qui se trouve à droite du curseur. S'il est situé au début de la ligne, celle-ci sera donc totalement supprimée.

Ctrl + W supprime le premier mot situé à gauche du curseur. Un « mot » est séparé par des espaces ; on s'en sert en général pour supprimer le paramètre situé à gauche du curseur.

Ctrl + Y si vous avez supprimé du texte avec une des commandes Ctrl + U, Ctrl + K ou Ctrl + W qu'on vient de voir, alors le raccourci Ctrl + Y « collera » le texte que vous venez de supprimer. C'est donc un peu comme un couper-coller.

CHAPITRE 2

Navigation dans les dossiers

pwd savoir où je suis

which cmd permet de localiser l'emplacement d'une commande cmd

ls -F rajoute à la fin des éléments un symbole pour qu'on puise faire la distinction entre les dossiers, fichiers, raccourcis…

ls -lh afficher la taille des éléments (-h : « for humans »)

cd sans arguments, ramène dans home/currentUser/

du pour Disk Usage (utilisation du disque) vous donne des informations sur la taille qu'occupent les dossiers sur votre disque

du -c produces a grand total

du -hsc * This finds the size recursively and puts it next to each folder name, along with total size at the bottom, all in the human format.

du -hs * | sort -h same as previous but sort by size

CHAPITRE 3

Manipuler les fichiers

`tree --du -h` returns a tree-like representation of the current folder and its subfilders/files (could be a huge dump)

Note that tree might have to be installed first with `sudo apt-get install tree`

`less -N fichier.txt` affiche le contenu d'un fichier avec numéro de lignes

`touch fichier.txt` crée un fichier vide

`head/tail` affiche le début/fin d'un fichier

`tail -f fichier` permet de voir l'évolution de la fin d'un fichier (par ex : var/log/syslog)

`.` le dossier où je me trouve

`cp -R dossierSource dossierDest` Avec l'option `-R` (un « R » majuscule !), vous pouvez copier un dossier, ainsi que tous les sous-dossiers et fichiers qu'il contient !

`mv` déplacer

`rm fichier` suprimer

`rm -i fichier` supprimer avec confirmation

`rm -f fichier` forcer la suppression quoi qu'il arrive

`rm -v fichier` supprimer avec verbose

`rm - r dossier/` supprimer un dossier et son contenu

`rm -rf *` supprimer tous les fichiers du dossier courant

ATTENTION : NE SURTOUT PAS CONFONDRE AVEC `rm -rf /*` qui supprime tout, partout et flingue tout le système

`ln fichier1 fichier2` crée un lien physique entre fichier1 et fichier2 (même « inode », pointent vers le même contenu)

`ln -s fichier1 fichier2` crée un lien symbolique entre fichier1 et fichier2 (~raccourci Windows)

`mkdir nom_du_dossier` créer un dossier

`mkdir -p dossier/{dossier1,dossier2,dossier3}` crée l'arborescence suivante :

```
├── [eric     4.0K]  dossier
│   ├── [eric     4.0K]  dossier1
│   ├── [eric     4.0K]  dossier2
│   └── [eric     4.0K]  dossier3
```

`-p`, `--parents` : no error if existing, make parent directories as needed

CHAPITRE 4

Les flux de redirection

`cmd > fichier.txt` redirige la sortie standard de la commande cmd vers le fichier fichier.txt

`cmd >> fichier.txt` redirige la sortie standard de la commande cmd vers la fin du fichier fichier.txt

`cmd > fichier.txt 2> erreurs.txt` redirige la sortie standard de la commande cmd vers le fichier fichier.txt et la sortie des erreurs vers le fichier erreurs.txt (fin du fichier : 2>>)

`cmd > fichier.txt 2>$1` redirige la sortie standard ET celle des erreurs de la commande cmd vers le fichier fichier.txt

`cmd < fichier.txt` envoie le contenu de fichier.txt à la commande cmd

`cmd << motDeFin` passe la console en mode saisie au clavier, ligne par ligne. Toutes ces lignes seront envoyées à la commande lorsque le mot-clé de fin « motDeFin » aura été écrit.

`cmd1 | cmd2` exécute cmd1 et envoie sa sortie en entrée de cmd2 (« pipe »)

`cmd > /dev/null 2>&1` démarre le programme sans récupérer sa sortie ni ses erreurs, qui sont « jetées » dans /dev/null

CHAPITRE 5

Surveiller l'activité du système

w permet de voir qui fait quoi

`USER` : le nom de l'utilisateur (son login) ;

`TTY` : le nom de la console dans laquelle se trouve l'utilisateur. Souvenez-vous que sous Linux il y a en général six consoles (tty1 à tty6) et qu'en plus de ça, on peut en ouvrir une infinité grâce aux consoles graphiques (leur nom commence par pts, en général), comme le propose le programme « Terminal » sous Unity ou « Konsole » sous KDE ;

`FROM` : c'est l'adresse IP (ou le nom d'hôte) depuis laquelle il se connecte. Ici, comme je me suis connecté en local (sur ma propre machine, sans passer par Internet), il n'y a pas vraiment d'IP ;

`LOGIN@` : l'heure à laquelle cet utilisateur s'est connecté ;

`IDLE` : depuis combien de temps cet utilisateur est inactif (depuis combien de temps il n'a pas lancé de commande)

`WHAT` : la commande qu'il est en train d'exécuter en ce moment. En général, si vous voyez bash, cela signifie que l'invite de commandes est ouverte et qu'aucune commande particulière n'est exécutée.

`ps` : liste statique des processus

ps -ef lister tous les processus (~ps -A)

ps -ejH afficher les processus en arbre

ps -u UTILISATEUR lister les processus lancés par un utilisateur

top liste dynamique des processus

`q` : ferme top ;

`h` : affiche l'aide, et donc la liste des touches utilisables.

`B` : met en gras certains éléments.

`f` : ajoute ou supprime des colonnes dans la liste.

`F` : change la colonne selon laquelle les processus sont triés. En général, laisser le tri par défaut en fonction de %CPU est suffisant.

`u` : filtre en fonction de l'utilisateur que vous voulez.

`k` : tue un processus, c'est-à-dire arrête ce processus. Ne vous inquiétez pas, en général les processus ne souffrent pas. On vous demandera le numéro (PID) du processus que vous voulez tuer. Nous reviendrons sur l'arrêt des processus un peu plus loin.

`s` : change l'intervalle de temps entre chaque rafraîchissement de la liste (par défaut, c'est toutes les trois secondes).

`kill processPid` tue le processus dont le PID est processPid (proprement)

`kill -9 processPid` force l'arrêt immédiat du processus processPid (bourrin)

`killall processName` tue plusieurs processus dont le com est processName

`sudo halt/reboot` arrête/reboot le PC

CHAPITRE 6

Exécuter des programmes en arrière-plan

`cmd &` démarre la commande cmd en arrière plan

`cmd > /dev/null 2>&1 &` démarre le programme en arrière plan sans récupérer sa sortie ni ses erreurs, qui sont « jetées » dans /dev/null

`nohup cmd` détacher le processus de la console (ave les autres méthodes le programme se ferme quand on ferme la console, là pas)

`Ctrl + z` mettre en pause l'exécution du programme

`bg` fait passer en arrière plan le processus que l'on a stoppé avec Ctrl+z

`jobs` affiche les processus en arrière plan

`fg` reprendre un processus au premier plan (foreground)

`fg %2` reprendre le processus n°2 (trouvé avec la commande jobs) en premier plan

`screen` permet d'avoir plusieurs consoles en une (installer : sudo apt-get install screen)

`Ctrl + a puis c` : créer une nouvelle « fenêtre ».

`Ctrl + a puis w` : afficher la liste des « fenêtres » actuellement ouvertes. En bas de l'écran vous verrez par exemple apparaître : 0-$ bash 1*$ bash. Cela signifie que vous avez deux fenêtres ouvertes, l'une numérotée 0, l'autre 1. Celle sur laquelle vous vous trouvez actuellement contient une étoile * (on se trouve donc ici dans la fenêtre n° 1).

`Ctrl + a puis A` : renommer la fenêtre actuelle. Ce nom apparaît lorsque vous affichez la liste des fenêtres avec Ctrl + a puis w.

`Ctrl + a puis n` : passer à la fenêtre suivante (next).

`Ctrl + a puis p` : passer à la fenêtre précédente (previous).

`Ctrl + a puis Ctrl + a` : revenir à la dernière fenêtre utilisée.

`Ctrl + a puis un chiffre de 0 à 9` : passer à la fenêtre n° X.

`Ctrl + a puis "` : choisir la fenêtre dans laquelle on veut aller.

`Ctrl + a puis k` : fermer la fenêtre actuelle (kill).

`Ctrl + a puis S` : découper screen en plusieurs parties (split)

`Ctrl + a puis d` : détache screen et vous permet de retrouver l'invite de commandes « normale » sans arrêter screen. C'est peut-être une des fonctionnalités les plus utiles que nous devons approfondir, et cela nous ramène d'ailleurs à l'exécution de programmes en arrière-plan dont nous avons parlé au début du chapitre.

`screen -r` récupérer son ancienne session screen (détachée)

`screen -ls` affiche la liste des screens actuellement ouverts

CHAPITRE 7

Exécuter un programme à une heure différée

`date` afficher / régler l'heure

`at HH:MM` donne la possibilité de démarrer un programme à HH :MM. Un promppt aparait pour demander lequel.

`at HH:MM tomorrow`

`at HH:MM 11/15/10` attention date au format américain

`at now +5 minutes`

`atq` afficher les jobs en attente -> donne un n°

`atrm x` supprimer un job en attente dont le numéro est x

`sleep x` attend x secondes (minute : xm, heure : xh, jour : xd)

`echo "export EDITOR=nano" >> ~/.bashrc` faire de Nano l'éditeur par défaut

`crontab -e` modifier la crontab

`crontab -l` afficher la crontab

`crontab -r` supprimer la crontab (immédiate et sans confirmation)

`m h dom mon dow command`

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

ex : `47 15 * * * touch /home/mateo21/fichier.txt` -> tous les jours à 15h47

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

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

CHAPITRE 8

Extraire, trier et filtrer des données

`grep -i mot fichier` affiches les occurences de « mot » dans fichier sans faire attention à la casse.

`grep -n mot fichier` affiche les n° de ligne

`grep -v mot fichier` inversion de la recherche : « tout ce qui ne contient PAS mot »

`grep -r mot répertoire` rechercher dans tous les fichiers et sous-dossiers (équivalent à rgrep)

```
grep "ma phrase contient des espaces" monFichier
```

`grep -E mot fichier` grep avec expression régulière

Expressions régulières :

- `.` : Caractère quelconque
- `^` : Début de ligne (cherche un mot placé en début de ligne)
- `$` : Fin de ligne (cherche un mot placé en fin de ligne)
- `[]` : Un des caractères entre les crochets
- `?` : L'élément précédent est optionnel (peut être présent 0 ou 1 fois)
- `*` : L'élément précédent peut être présent 0, 1 ou plusieurs fois
- `+` : L'élément précédent doit être présent 1 ou plusieurs fois
- `|` : Ou
- `()` : Groupement d'expressions

`sort fichier` trier le contenu d'un fichier

`sort -o noms_tries.txt noms.txt` avec sortie vers noms_tries.txt

`sort -R fichier` trier aléatoirement

`sort -n fichier` trier des nombres (ne se bas pas sur l'alphabet, sinon : 1 123 23 . . .)

`wc fichier.txt`

renvoie un résultat type « a b c fichier.txt » où :

- `a` : nb de lignes (`-l`)
- `b` : nb de mots (`-w`)
- `c` : nb d'octets (`-c`)

`wc -m fichier.txt` nb de caractères dans le fichier

`uniq fichier.txt` supprime les doublons

`uniq doublons.txt sans_doublons.txt` sort ça dans sans_doublons.txt

uniq -c compte le nb d'occurences

ex : `uniq -c doublons.txt`

résultats :

```
1 Albert
3 François
1 Jean
2 Marcel
```

uniq -d fichier uniquement les lignes en double

cut -c 2-5 noms.txt conserve uniquement les caractères 2 à 5 de chaque ligne

cut -c -3 noms.txt conserve uniquement les caractères 1 à 3 de chaque ligne

cut -c 3- noms.txt du n°3 au dernier de chaque ligne

cut -d , -f 1 notes.csv

- `-d` : indique quel est le délimiteur dans le fichier (ici ",")
- `-f` : indique le numéro du ou des champs à couper, cad que l'on garde (ici le 1er)

cut -d , -f 1,3 notes.csv garde les champs 1 ET 3

cut -d , -f 1-3 notes.csv garde les champs 1 à 3

CHAPITRE 9

Les utilisateurs et les droits

`adduser userName` ajouter un utilisateur userName

`deluser userName` supprimer un utilisateur userName

`deluser --remove-home userName` supprime aussi son répertoire personnel

`pswd userName` changer le mot de passe de userName

`addgroup groupName` crée un groupe d'utilisateurs

`usermod -l userName` modifier le nom de l'utilisateur

`usermod -g groupName userName` modifier le groupe d'un utilisateur (remplace les précédents groupes)

`usermod -G amis,paris,collegues patrick` ajouter l'utilisateur à plusieurs groupes (G majuscule) (remplace les précédents groupes)

`usermod -aG amis,paris,collegues patrick` ajoute l'utilisateur aux groupes en gardant ses groupes précédents

`delgroup groupName` supprime un groupe

`chown userName fichier` rend patrick propriétaire de rapport.txt

`chgrp groupName fichier` rend le groupe groupName propriétaire du fichier

`chown userName:groupName fichier` Cela affectera le fichier à l'utilisateur userName et au groupe groupName.

`chown -R userName:userName /dossier/` modifie tous les sous-dossiers et fichiers contenus dans un dossier pour y affecter un nouvel utilisateur (et un nouveau groupe si on utilise la technique du deux points)

ex [`drwxr-xr-x 2 mateo21 mateo21 4096 2007-11-13 21:53 Desktop`]

- `d` (Directory) : indique si l'élément est un dossier ;
- `l` (Link) : indique si l'élément est un lien (raccourci) ;
- `r` (Read) : indique si on peut lire l'élément ;
- `w` (Write) : indique si on peut modifier l'élément ;
- `x` (eXecute) : si c'est un fichier, « x » indique qu'on peut l'exécuter. Ce n'est utile que pour les fichiers exécutables (programmes et scripts).

Si c'est un dossier, « x » indique qu'on peut le « traverser », c'est-à-dire qu'on peut voir les sous-dossiers qu'il contient si on a le droit de lecture dessus.

- le premier triplet rwx indique les droits que possède le propriétaire du fichier sur ce dernier ;

— le second triplet rwx indique les droits que possèdent les autres membres du groupe sur ce fichier ;
— enfin, le dernier triplet rwx indique les droits que possèdent tous les autres utilisateurs de la machine sur le fichier.

chmod

modifie les droits -> utiliser des numéros :

— r = 4
— w = 2
— x = 1
donc :

— --- = 0+0+0 = 0
— r-- = 4+0+0 = 4
— -w- = 0+2+0 = 2
— --x = 0+0+1 = 1
— rw- = 4+2+0 = 6
— -wx = 0+2+1 = 3
— r-x = 4+0+1 = 5
— rwx = 4+2+1 = 7

chmod 600 rapport.txt -rw------- 1 mateo21 mateo21 0 2007-11-15 23:14 rapport.txt

CHAPITRE 10

Nano : éditeur de texte

nano -m autorise la souris

nano -i indentation autorise

nano -A active le retour intelligent au début de la ligne. Normalement, lorsque vous appuyez sur la touche Origine(aussi connue sous le nom de Home) située à côté de la touche Fin, le curseur se repositionne au tout début de la ligne. Avec cette commande, il se positionnera après les alinéas.

Alt + Shift + 6 Copier une ligne

Options par défaut : ouvrir `.nanosrc`

```
set mouse
    set autoindent
    set smarthome
```

CHAPITRE 11

Alias

créer un alias dans `.bashrc` : ex : `alias ll='ls -l'`

CHAPITRE 12

Installer des programmes avec apt-get

`/etc/apt/sources.list` :
- — adresses des serveurs pour les dépôts
- — pour autoriser le téléchargement de sources

ex :

```
deb-src http://raspbian.raspberrypi.org/raspbian/ stretch main contrib non-free rpi
                                                  ^ OS version
```

`apt-get update` (optionnel) pour mettre notre cache à jour si ce n'est pas déjà fait

`apt-cache search monpaquet` (optionnel) pour rechercher le paquet que nous voulons télécharger si nous ne connaissons pas son nom exact

`apt-get install monpaquet` pour télécharger et installer notre paquet

`apt-cache show nomdupaquet` pour une plus ample description d'un paquet

`apt-get remove nomdupaquet` désinstaller un paquet. Toutefois, cela ne supprime pas les dépendances du paquet devenues inutiles

`apt-get autoremove nomdupaquet` Pour demander à `apt-get` de supprimer aussi les dépendances inutiles, on utilise `autoremove`

`apt-get upgrade` met à jour tous les paquets du système d'un seul couper (faire un `apt-get update` avant)

CHAPITRE 13

A propos des manuels

man
— `/ + "..." + Entrée` : faire une recherche. Résultat suivant : à nouveau taper /

synopsis dans le man :
— gras : tapez le mot exactement comme indiqué
— souligné : remplacez le mot souligné par la valeur qui convient dans votre cas
— `[-hvc]` : toutes les options -h, -v et -c sont facultatives
— `a|b` : vous pouvez écrire l'option « a » OU « b », mais pas les deux à la fois
— `option...` : les points de suspension indiquent que l'option peut être répétée autant de fois que vous voulez

apropos quelqueChose va rechercher toutes les commandes qui parlent de quelqueChose dans leur page du manuel.

cmd -h si implémenté par la commande, affiche une aide résumée

whatis cmd donne juste l'en-tête du manuel pour expliquer en deux mots à quoi sert la commande

CHAPITRE 14

Rechercher des fichiers

`locate fichier` recherche les fichiers/dossiers qui contiennent « fichier » et qui sont dans la base de données (db)

`sudo updatedb` mettre à jour la base de données

`find fichier` recherche des fichiers en scannant le disque dur (pas la db base)

`find -name "file.txt"` dans le dossier courant

`find /var/log/ -name "syslog"` dans un dossier différent du courant

`find -size +10M` chercher les fichiers qui font plus de 10Mo

`find -name "*.odt" -atime -6` chercher les fichiers .odt auxquels on a accédé depuis 7 jours (numérotation comence à 0)

`find -type d` only directories

`find -type f` only files

`find -name "*.jpg" -delete` supprime tous les fichiers .jpg (dangereux ! ! !)

`find -name ".jpg" -exec cmd {} \;` exécute la commande cmd avec les résultats. La commande doit finir par un `\;` obligatoirement

CHAPITRE 15

Archiver et compresser

fichier `tar` : archive permettant de regrouper plusieurs fichiers

gzip/bzip2 [pour compresser des fichiers (des tar par exemple)]
- `gzip` : c'est le plus connu et le plus utilisé ;
- `bzip2` : il est un peu moins fréquemment utilisé. Il compresse mieux mais plus lentement que `gzip`.

tar -cvf nom_archive.tar nom_dossier/ créer une archive
- `-c` : signifie créer une archive tar
- `-v` : signifie afficher le détail des opérations
- `-f` : signifie assembler l'archive dans un fichier.

15.1 Archiver

tar -tf archive.tar afficher le contenu de l'archive sans l'extraire

tar -rvf archive.tar fichier_supplementaire ajouter un fichier à l'archive

tar -xvf archive.tar extraire les fichiers de l'archive

15.2 Compresser une archive

gzip archive.tar compresse l'archive avec gzip -> ajoute un .gz à la fin (archive.tar.gz)

gunzip archive.tar.gz décompresse l'archive

bzip2 archive.tar idem avec `bzip2`

bunzip2 archive.tar.bz2 idem avec `bzip2`

15.3 Archiver et compresser en une commande

tar -zcvf archive.tar.gz tutoriels/ archive et compresse en une comande (ici avec `gzip`)

tar -zxvf archive.tar.gz décompresse et désarchive en une commande

tar -jcvf tutoriels.tar.bz2 tutoriels/ idem avec `bzip2`

tar -jxvf tutoriels.tar.bz2 tutoriels/ idem avec `bzip2`

`tar -jxvf tutoriels.tar.bz2 tutoriels/`
idem avec `bzip2`

15.4 Afficher un fichier archivé sans le désarchiver

`zcat`, `zmore` & `zless`
afficher directement un fichier compressé (fichier simple, pas archive)

15.5 Fichiers `.zip`

`sudo apt-get install unzip`
installer le décompresseur de zip

`unzip archive.zip`
décompresser un zip

`unzip -l fichier.zip`
afficher le contenu du fichier zip sans l'extraire

`sudo apt-get install zip`
installer le compresseur de zip

`zip -r tutoriels.zip tutoriels/`
Le -r demande à compresser tous les fichiers contenus dans le dossier tutoriels (sans ce paramètre, seul le dossier, vide, sera compressé !).

15.6 Fichiers `.rar`

`sudo apt-get install unrar`

`unrar e tutoriels.rar` ' Non, vous ne rêvez pas, l'auteur du programme ne veut pas que l'on mette un tiret devant l'option e ! Il faut bien qu'il y ait des exceptions dans la vie. :-)

unrar l tutoriels.rar Pour lister le contenu avant décompression, utilisez l'option

pas possible de créer des fichiers .rar (format propriétaire)

CHAPITRE 16

La connexion sécurisée à distance avec SSH

— Telnet : non sécurisé (non crypté)
— SSH : ~Telnet crypté

`ssh rico@192.168.1.5` se connecte en ssh au login `rico` et à la machine d'adresse ip `192.168.1.5`

CHAPITRE 17

Transférer des fichiers

17.1 `wget` : télécharger un fichier sur le web

`wget http://website.com/file` télécharger un fichier distant

17.2 `scp` : copier/coller un fichier par SSH

`scp fichier_origine copie_destination` permet de copier des fichiers distants de manière sécurisée (cryptage ssh)

`scp fichier remoteLogin@85.123.10.201:/home/remoteLogin/dossier/` copier un fichier de l'ordi local vers le distant

`scp remoteLogin@85.123.10.201:fichier copie_fichier` pas nécessaire de préciser le nom -> gardera le même nom que le fichier d'origine

`scp -P 16296 mateo21@85.123.10.201:image.png` en précisant le port (attention MAJUSCULE)

17.3 Protocole ftp non sécurisé

`ftp://ftp.debian.org` connection au serveur ftp de debian (login : anonymous mpd : any)

- on a ensuite un prompt qui nous permet de naviguer sur le serveur (`ls`, `cd`, `pwd`...)
- `put` : ajouter un fichier sur le serveur (verouillé dans le cas de celui de debian)
- `get` : récupérer un fichier depuis le serveur (sera mis dans le dossier courant du pc local)
- pour se déplacer dans le pc local : `!cd`, `!ls`, `!pwd`... (ajouter un ! avant la commande)
- attention : protocole ftp pas sécurisée

17.4 Protocole ftp sécurisé

`sftp mateo21@lisa.simple-it.fr` ftp sécurisée avec ssh (port par défaut : 22)

17.5 `rsync` : sauvegardes sur un serveur distant

`rsync` permet de créer des sauvegardes sur un serveur distant (incrémentielles, etc. . .)

`rsync -arv Images/ backups/` analyse les différences entre /Images et /backup et fait une sauvegarde

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

`rsync -arv --delete Images/ backups/` analyse les différences entre /Images et /backup et efface les fichiers de backups qui ne sont plus dans /Images

`rsync -arv --delete --backup Images/ backups/` garde les fichiers suprimés en leur ajoutant un suffixe dans le dossier de sauvegarde

`rsync -arv --delete --backup --backup-dir=/home/mateo21/backups_supprimes Images/ backup`
les fichiers suprimés vont dans le dossier /home/mateo21/backups_supprimes

`rsync -arv --delete --backup --backup-dir=/home/mateo21/fichiers_supprimes Images/ mateo`
fait le backup sur un ordinateur distant via ssh

`rsync -arv --delete --backup --backup-dir=/home/mateo21/fichiers_supprimes Images/ mateo`
avec un n° de port custom

CHAPITRE 18

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

`host www.google.com` renvoie l'adresse ip du site

pour voir les associations (~DNS) locales : -> /etc/hosts

`whois google.com` obtenir des infos sur un site (installer d'abord le package whois)

`ifconfig` liste des interfaces réseau `lo` : c'est la boucle locale. Tout le monde devrait avoir cette interface. Elle correspond à une connexion à... vous-mêmes. C'est pour cela qu'on l'appelle la boucle locale : tout ce qui est envoyé par là vous revient automatiquement. Cela peut paraître inutile, mais on a parfois besoin de se connecter à soi-même pour des raisons pratiques.

`ifconfig eth0 down/up` activer/désactiver une interface

`netstat` statistiques sur le réseau

`netstat -i` statistiques des interfaces réseau

`netstat -uta` lister toutes les connexions ouvertes

Options :
- `-u` : afficher les connexions UDP ;
- `-t` : afficher les connexions TCP ;
- `-a` : afficher toutes les connexions quel que soit leur état.
- `-n` : affiche les n° de port plutôt que leur description
- `-l` : affiche les connexions en état d'écoute

Statuts :
- `ESTABLISHED` : la connexion a été établie avec l'ordinateur distant
- `TIME_WAIT` : la connexion attend le traitement de tous les paquets encore sur le réseau avant de commencer la fermeture
- `CLOSE_WAIT` : le serveur distant a arrêté la connexion de lui-même (peut-être parce que vous êtes restés inactifs trop longtemps ?)
- `CLOSED` : la connexion n'est pas utilisée
- `CLOSING` : la fermeture de la connexion est entamée mais toutes les données n'ont pas encore été envoyées
- `LISTEN` : à l'écoute des connexions entrantes. Les connexions à l'état LISTEN ne sont pas utilisées actuellement mais qu'elles « écoutent » le réseau au cas où quelqu'un veuille se connecter à votre ordinateur

`netstat -s` statistiques résumées

`iptables -L` afficher les règles
- `Chain INPUT` : correspond aux règles manipulant le trafic entrant ;

- — `Chain FORWARD` : correspond aux règles manipulant la redirection du trafic ;
- — `Chain OUTPUT` : correspond aux règles manipulant le trafic sortant.
- — `target` : ce que fait la règle. Ici c'est ACCEPT, c'est-à-dire que cette ligne autorise un port et / ou une IP
- — `prot` : le protocole utilisé (tcp, udp, icmp). Je rappelle que TCP est celui auquel on a le plus recourt. ICMP permet à votre ordinateur de répondre aux requêtes de type « ping »
- — `source` : l'IP de source. Pour INPUT, la source est l'ordinateur distant qui se connecte à vous
- — `destination` : l'IP de destination. Pour OUTPUT, c'est l'ordinateur auquel on se connecte
- — la dernière colonne : elle indique le port après les deux points « : ». Ce port est affiché en toutes lettres, mais avec -n vous pouvez obtenir le numéro correspondant

`iptables -F` réinitialise les règles du pare-feu (attention : efface tout le travail fait auparavant sur le pare-feu...)

Ajouter et supprimer des règles :

Voici les principales commandes à connaître :

- — `-A chain` : ajoute une règle en fin de liste pour la chain indiquée (INPUT ou OUTPUT, par exemple).
- — `-D chain rulenum` : supprime la règle n° rulenum pour la chain indiquée.
- — `-I chain rulenum` : insère une règle au milieu de la liste à la position indiquée par rulenum. Si vous n'indiquez pas de position rulenum, la règle sera insérée en premier, tout en haut dans la liste.
- — `-R chain rulenum` : remplace la règle n° rulenum dans la chain indiquée.
- — `-L` : liste les règles (nous l'avons déjà vu).
- — `-F chain` : vide toutes les règles de la chain indiquée. Cela revient à supprimer toutes les règles une par une pour cette chain.
- — `-P chain regle` : modifie la règle par défaut pour la chain. Cela permet de dire, par exemple, que par défaut tous les ports sont fermés, sauf ceux que l'on a indiqués dans les règles.
- — `-m (--match)` : Specifies a match to use, that is, an extension module that tests for a specific property. The set of matches make up the condition under which a target is invoked. Matches are evaluated first to last as specified on the command line and work in short-circuit fashion, i.e. if one extension yields false, evaluation will stop.

`iptables -A (chain) -p (protocole) --dport (port) -j (décision)` ajouter une règle. Remplacez chain par la section qui vous intéresse (`INPUT` ou `OUTPUT`), protocole par le nom du protocole à filtrer (TCP, UDP, ICMP...) et enfin décision par la décision à prendre : `ACCEPT` pour accepter le paquet, `REJECT` pour le rejeter ou bien `DROP` pour l'ignorer complètement.

exemple : `iptables -A INPUT -p tcp --dport ssh -j ACCEPT`

`iptables -A INPUT -p icmp -j ACCEPT` autoriser les pings (protocole ICMP)

```
iptables -A INPUT -i lo -j ACCEPT
```

```
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

- — La première règle autorise tout le trafic sur l'interface de loopback locale grâce à -i lo. Il n'y a pas de risque à autoriser votre ordinateur à communiquer avec lui-même, d'autant plus qu'il en a parfois besoin !
- — La seconde règle autorise toutes les connexions qui sont déjà à l'état ESTABLISHED ou RELATED. En clair, elle autorise toutes les connexions qui ont été demandées par votre PC. Là encore, cela permet d'assouplir le pare-feu et de le rendre fonctionnel pour une utilisation quotidienne.

`iptables -P INPUT DROP` on refuse tout ce qui n'est pas aurtorisé (faire de même pour OUTPUT)

ATTENTION : ces règles seront perdues au redémarrage ! pour que ça ne soit pas le cas -> ajouter un script shell

CHAPITRE 19

Compiler un programme depuis les sources

fichier `.deb` : pour installer un programme sur les dérivées de Debian (Red Hat : `.rpm`)

Convertir un `.rpm` en `.deb` : utiliser le programme `alien`

`dpkg -i foo.deb` installer un package `foo.deb`

19.1 Pour compiler un programme depuis une source

1. **Installer build-ensential (si pas encore installé)** `sudo apt-get install build-essential`
2. **Récupérer l'archive contenant le programme et la décompresse/désarchiver :** `tar zxvf htop-0.8.3.tar.gz` (ici `htop-0.8.3`)
3. **Exécuter `./configure` pour vérifier les dépendances** `./configure`
4. Installer ce qui manque au fur et à mesure (chercher sur google pour trouver le nom du paquet à installer pour les headers manquants par exemple) et relancer `./configure` jusqu'à ce qu'il n'y ait plus d'erreurs
5. **Lancer la compilation** `make` (attention à être dans le répertoire de la source)
6. **A la fin de la compilation un éxécutable a été créé. Il ne reste plus qu'à l'installer, c'est-à-dire à le copier dans le bon répe** `sudo make install`
7. **Le programme est installé ! Pour le lancer :** `htop` (dans le cas de notre exemple)
8. On peut maintenant supprimer le répertoire contenant les fichiers source

CHAPITRE 20

VIM : éditeur de texte avancé

`vim` lancer vim

`vimtutor` lancer le tutoriel vim

20.1 Modes de VIM

— **Mode interactif** : par défaut. Permet de se déplacer dans le texte, de supprimer une ligne, copier-coller du texte, rejoindre une ligne précise, annuler ses actions, etc
— **Mode insertion** : appuyer sur `i` pour y entrer
— **Mode commande** : l'activer en tapant **`:`**

Racourcis	
`i`	insérer du texte
`ESC`	sortir du mode insertion

Commandes	
`:w`	enregistrer le fichier (write)
`:q`	quitter
`:wq`	enregistrer puis quitter

Déplacements	
`h`	gauche
`j`	bas
`k`	haut
`l`	droite
`0`	aller en début de ligne (origin)
`$`	aller en fin de ligne
`w`	se déplacer de mot en mot (word)

```
      ^
      k
< h       l >
      j
      v
```

Opérations standards	
`x`	effacer des lettres en mode interactif
`(nb)x`	effacer nb lettres
`dw`	effacer un mot
`dd`	effacer une ligne
`d0`	supprimer du curseur au début de la ligne
`d$`	supprimer du curseur à la fin de la ligne
`yy`	copier une ligne en mémoire
`p`	coller (coller plusieurs fois : ex : 8p -> 8x)
`r`	remplacer une lettre
`u`	annuler des modifications
`G`	aller à la ligne x (Go)

Opérations avancées	
`/`	passer en mode recherche (pour chercher un mot par ex) `n` aller à la prochaine occurence `N` aller à la précédente occurence
`:s/ancien/nouveau`	remplacer le mot « ancien » par le mot « nouveau » :
`:s/ancien/nouveau`	remplace la première occurrence de la ligne où se trouve le curseur ;
`:s/ancien/nouveau/g`	remplace toutes les occurrences de la ligne où se trouve le curseur ;
`:#,#s/ancien/nouveau/g`	remplace toutes les occurrences dans les lignes n° # à # du fichier ;
`:%s/ancien/nouveau/g`	remplace toutes les occurrences dans tout le fichier. C'est peut-être ce que vous utiliserez le plus fréquemment.
`:r`	fusion de fichiers : insérer le contenu d'un fichier au curseur

Splitter écrans (viewports)	
`:sp`	découper l'écran horizontalement
`:sp autrefichier`	ouvrir autrefichier dans la seconde moitié de l'écran
`:vsp`	découper l'écran verticalement
`Ctrl + w` puis `Ctrl + ``w`	navigue de viewport en viewport. Répétez l'opération plusieurs fois pour accéder au viewport désiré.
`Ctrl + w` puis `j`	déplace le curseur pour aller au viewport juste en dessous. La même chose fonctionne avec les touches `h`, `k`, `j` et `l` que l'on utilise traditionnellement pour se déplacer dans Vim.
`Ctrl + w` puis +	agrandit le viewport actuel.
`Ctrl + w` puis −	réduit le viewport actuel.
`Ctrl + w` puis =	égalise à nouveau la taille des viewports.
`Ctrl + w` puis `r`	échange la position des viewports. Fonctionne aussi avec « R » majuscule pour échanger en sens inverse.
`Ctrl + w` puis `q`	ferme le viewport actuel.

Zoom	
`Ctrl + Shift + +`	zoom
`Ctrl + Shift + -`	dezoom

20.2 Options de vim

Rem : pour qu'elles soient retenues, créer un fichier .vimrc dans le répertoire personnel (un exemple de fichier se trouve dans /etc/vim -> vimrc)

`:set option`	activer l'option en mode commande
`:set nooption`	désactiver l'option en mode commande
`:set option?`	connaitre l'état d'une option
`:set option=valeur`	donner une valeur à une option
`:set syntax=ON`	coloration synthaxique
`:set background=dark`	coloration adaptée pour les fonds noirs
`:set number`	affiche les n° de lignes
`:set showcmd`	afficher la commande en cours
`:set ignorecase`	ignorer la casse lors des recherches
`:set mouse=a`	activer la souris

20.3 Gérer les plugins

https://artisan.karma-lab.net/configurer-vim

— Création d'une arborescence pour les fichiers de config :

```
$ cd
$ mkdir -p .vim/{autoload,colors,syntax,plugin,spell,config}
$ mv .vimrc .vim/vimrc
$ ln -s .vim/vimrc .vimrc
```

(on crée un lien vers `.vim/vimrc`)

— Installation de `pathogen` :

```
$ cd ~/.vim
$ git clone https://github.com/tpope/vim-pathogen.git pathogen
$ cd autoload
$ ln -s ../pathogen/autoload/pathogen.vim
```

— Pour mettre à jour `pathogen` :

```
$ cd ~/.vim/pathogen
$ git pull
```

— Installer un plugin : exemple avec `NERDTree`

```
$ cd ~/.vim
$ mkdir -p bundle
$ cd bundle
$ git clone https://github.com/scrooloose/nerdtree.git nerdtree
```

— Allure finale du dossier `.vim` (situé dans `~/`)

```
.vim
├── autoload
│   └── pathogen.vim -> ../pathogen/autoload/pathogen.vim
├── bundle
│   └── nerdtree
│       ├──.......
├── colors
├── config
├── pathogen
│   ├── autoload
│   │   └── pathogen.vim
│   ├── CONTRIBUTING.markdown
│   └── README.markdown
├── plugin
├── spell
├── syntax
└── vimrc (fichier)
```

— Allure du fichier `vimrc` :

```
1 set nocompatible
2
3 runtime! config/**/*.vim
4
5 set number
```

(suite sur la page suivante)

(suite de la page précédente)

```
6
7 " Initialisation de pathogen
8 call pathogen#infect()
9 call pathogen#helptags()
10
```

— Pour démarrer NerdTree : taper `:NERDTree` en mode interactif

CHAPITRE 21

Scripts Shell

21.1 Remarques générales

— ici on utilisera bash
— Extension : `fichier.sh`
— ajouter `#!/bin/bash` au début pour s'assurer qu'il est éxécuté avec bash et pas un autre shell
— comentaires : `#`
— **donner le droit « éxécutable » au script :** `chmod +x essai.sh`
— **Exécuter le script :** `./essai.sh`

— **Mode debug** [`bash -x`] `bash -x essai.sh`
— **Pour éxécuter un script à n'importe quel endroit -> copier le script dans un des répertoires du `PATH`**
`echo $PATH` pour voir où se trouvent les répertoires du PATH

21.2 Variables

Attention pas d'espaces autour des "="

`echo something` renvoie le paramètre something
`echo $something` pour afficher une variable dans un script (ajouter "$")
`echo -e "test\ntes2\n"` tient comptre des retours à la ligne `\n`

21.3 Quotes

— **Simple quotes** : bash ne tient pas compte de la variable quand on utilise des simple quotes `'...'`

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

— **Double quotes** : bash tient compte de la variable quand on utilise des double quotes `"..."`

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

— **Back quotes** : bash éxécute ce qui se trouve dans les back quotes `` `...` ``

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

21.4 `read` : Demander une saisie

— `-p` : demande un message de prompt

```
$ read -p 'Entrez votre nom : ' nom prenom
$ echo "Bonjour $nom $prenom !"
Entrez votre nom : Mathieu
Bonjour Deschamps Mathieu !
```

— `-n` : nb max de caractères

```
$ read -p 'Entrez votre login (5 caractères max) : ' -n 5 nom
$ echo "Bonjour $nom !"
```

— `-t` : précise un temps max

```
$ read -p 'Entrez le code de désamorçage de la bombe (vous avez 5 secondes) : '  -t 5␣
↪code
$ echo -e "\nBoum !"
```

— `-s` : ne pas afficher les caractères saisis

```
$ read -p 'Entrez votre mot de passe : ' -s pass
$ echo -e "\nMerci ! Je vais dire à tout le monde que votre mot de passe est $pass !␣
↪:)"
```

21.5 Opérations mathématiques

```
$ let "a = 5"
$ let "b = 2"
$ let "c = a + b"
7
```

Remarque : on peut utiliser : `let "a += 5"`

Opérations utilisables :

— l'addition : +
— la soustraction : -
— la multiplication : *
— la division : /
— la puissance : **
— le modulo (renvoie le reste de la division entière) : %

Opérations sur des nombres décimaux : cf commande `bc`

21.6 Les variables d'environnement

Elles sont disponibles tout le temps pour tous les scripts (variables globales) On les met en majuscules par convention.

Exemples :

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

Pour en créer de nouvelles : `export`

21.7 Les paramètres

`./script.sh param1 param2 param3`

Pour récupérer ces paramètres dans le script :

— `$#` : contient le nombre de paramètres
— `$0` : contient le nom du script exécuté (ici ./variables.sh)
— `$1` : contient le premier paramètre
— `$2` : contient le second paramètre
— ...
— `$9` : contient le 9e paramètre

Si on a plus de 9 variables : décaler l'ordre avec la commande `shift` . Exemple :

Script :

```
echo "Le paramètre 1 est $1"
shift
echo "Le paramètre 1 est maintenant $1"
```

A l'exécution

```
$ ./variables.sh param1 param2 param3
Le paramètre 1 est param1
Le paramètre 1 est maintenant param2
```

21.8 Les Tableaux

— **Définir un tableau :** `tableau=('valeur0' 'valeur1' 'valeur2')`
— **Accéder à une case particulière :** `${tableau[2]}`
— **Définir le contenu d'une case :** `tableau[2]='valeur2'`
— On peut sauter des cases :

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

— **Afficher l'ensemble d'une tableau :** `${tableau[*]}`

21.9 Les conditions

Synthaxe :

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

21.9.1 Tests sur les chaines de caractères

Rappel : toutes les variables sont traitées comme des strings

`$chaine1 = $chaine2` Vérifie si les deux chaînes sont identiques. Notez que bash est sensible à la casse : « b » est donc différent de « B ».
Il est aussi possible d'écrire == pour les habitués du langage C.

`$chaine1 != $chaine2` Vérifie si les deux chaînes sont différentes.

`-z $chaine` Vérifie si la chaîne est vide.

`-n $chaine` Vérifie si la chaîne est non vide.

21.9.2 Tests sur les nombres

$num1 -eq $num2 Vérifie si les nombres sont égaux (equal). À ne pas confondre avec le « = » qui, lui, compare deux chaînes de caractères.

$num1 -ne $num2 Vérifie si les nombres sont différents (nonequal).

Encore une fois, ne confondez pas avec ‘‘ !=‘‘ qui est censé être utilisé sur des chaînes de caractères.

$num1 -lt $num2 Vérifie si num1 est inférieur (<) à num2 (lowerthan).

$num1 -le $num2 Vérifie si num1 est inférieur ou égal (<=) à num2 (lowerorequal).

$num1 -gt $num2 Vérifie si num1 est supérieur (>) à num2 (greaterthan).

$num1 -ge $num2 Vérifie si num1 est supérieur ou égal (>=) à num2 (greaterorequal).

21.9.3 Tests sur les fichiers

-e $nomfichier Vérifie si le fichier existe.

-d $nomfichier Vérifie si le fichier est un répertoire. N'oubliez pas que sous Linux, tout est considéré comme un fichier, même un répertoire !

-f $nomfichier Vérifie si le fichier est un... fichier. Un vrai fichier cette fois, pas un dossier.

-L $nomfichier Vérifie si le fichier est un lien symbolique (raccourci).

-r $nomfichier Vérifie si le fichier est lisible (r).

-w $nomfichier Vérifie si le fichier est modifiable (w).

-x $nomfichier Vérifie si le fichier est exécutable (x).

$fichier1 -nt $fichier2 Vérifie si fichier1 est plus récent que fichier2 (newerthan).

$fichier1 -ot $fichier2 Vérifie si fichier1 est plus vieux que fichier2 (olderthan).

21.9.4 Plusieurs tests : && et ||

Exemple :

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

21.9.5 Test inversé : not

Exemple :

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

21.10 Switch case

Exemple :

```
case $1 in
        "Bruno")
                echo "Salut Bruno !"
                echo "tu vas bien ?"
                ;;
        "Michel")
                echo "Bien le bonjour Michel"
                ;;
        "Jean")
                echo "Hé Jean, ça va ?"
                ;;
        *)
                echo "J'te connais pas, ouste !"
                ;;
esac
```

Exemple 2 : avec des ou (attention : `|` et pas `||`)

```
case $1 in
        "Chien" | "Chat" | "Souris")
                echo "C'est un mammifère"
                ;;
        "Moineau" | "Pigeon")
                echo "C'est un oiseau"
                ;;
        *)
                echo "Je ne sais pas ce que c'est"
                ;;
esac
```

21.11 Les boucles

21.11.1 Boucle `while`

Exemple :

```
# La réponse est vide ? Différente de "oui" ?
while [ -z $reponse ] || [ $reponse != 'oui' ]
do
        read -p 'Dites oui : ' reponse
done
```

21.11.2 Boucle `until`

`while` s'exécute tant que la condition est vraie, `until` jusqu'à ce qu'elle le soit

Exemple :

```
# Boucle jusqu'à ce que la réponse soit non vide et qu'elle soit "oui"
until [ -n $reponse ] && [ $reponse == 'oui' ]
do
        read -p 'Dites oui : ' reponse
done
```

21.11.3 Boucle `for`

Exemple :

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

Exemple 2 :

```
liste_fichiers=`ls`

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

Exemple 3 :

```
liste_fichiers=`ls`

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

Exemple 4 : renommer chacun des fichiers du répertoire actuel en leur ajoutant un suffixe -old par exemple

```
for fichier in `ls`
do
        mv $fichier $fichier-old
done
```

Exemple 5 : pour simuler une boucle for plus classique (seq génère tous les nombres allant du premier paramètre au dernier paramètre)

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

Remarque : pour aller de 2 en 2 : `seq 1 2 10`

21.12 Les fonctions

Déclarer une fonction : 2 possibilités

```
maFonction ()
{
        bloc d'instructions
}
```

ou :

```
function maFonction
{
        bloc d'instructions
}
```

Appeler une fonction :

```
maFonction      # tout simplement ;)
```

CHAPITRE 22

Indices and tables

— genindex
— modindex
— search