Commandes de base Unix/Linux

I. Arborescence de fichiers/répertoires

I.1. Commandes de base

Les commandes minimales à maîtriser sont :

- **`cd`** (changement de répertoire)
- **`pwd`** (affichage du répertoire courant)
- **`ls`** et options **`[-a -l -t -R -F]`** (liste des fichiers / répertoires)
- **`cp`** (*copie* de fichier.s / répertoire.s)
- **`mv`** (*déplacement* ou *renommage* de fichier.s / répertoire.s)
- **`rm`** (suppression de fichier.s) (**`shred`** : suppression "paranoïaque")
- **`mkdir`** (création de répertoire.s)
- **`rmdir`** (suppression de répertoire.s)
- **`cat`** (affichage du contenu de fichier.s)

Toutes les commandes de base Unix/Linux et bien d'autres ont un **manuel** en ligne de commande :

```
$ man <commande>
```

I.2. Chemin d'accès absolu ou relatif

Exécuter la commande `ls` en utilisant des **chemins d'accès absolus ou relatifs** :

```
$ which ls  # ou bien : where ls
/usr/bin/ls
```

Se placer dans / :

```
$ cd /usr
```

Que se passe-t-il si on exécute :

`$ ./bin/ls` ou `$ ../ls` ?

Se placer dans `/usr/lib` :

```
$ cd /usr/lib
```

Que se passe-t-il si on exécute :

`$ ../bin/ls` ?

- Créer un répertoire de travail **`info4108`** (**en minuscules SVP**) dans le dossier `Documents` de votre répertoire principal (*home directory* défini par la variable d'environnement **`$HOME`**) :

```
$ cd ~/Documents  # ou bien : cd $HOME/Documents
$ mkdir info4108
```

- Recopier **récursivement** le répertoire `/home/eric/public_html/a19/info4108/commandes` comme *sous-répertoire* du répertoire `info4108`. Le répertoire `commandes` contient entre autre un programme exécutable `somme`.
- Donner deux manières de lancer l'exécutable `somme` depuis votre répertoire principal ($HOME).
- Donner deux manières de lancer l'exécutable `somme` de l'utilisateur `eric` (dont le répertoire principal est `/home/eric`).

I.3. Copies (cp), déplacements, renommages (mv)

Les commandes `cp` et `mv` manipulent un ou plusieurs arguments sources et un argument destination et consistent respectivement à copier ou déplacer la ou les sources vers la destination.
Par exemple `cp fichier1 fichier2` recopie le contenu de `fichier1` dans un fichier nommé `fichier2`. Il existe plusieurs cas d'utilisation de ces commandes selon que :

- la source est un fichier existant ou non, est un répertoire existant ou non,
- la destination est un fichier existant ou non, est un répertoire existant ou non.

Par exemple si `fichier2` est le nom d'un **sous-répertoire** du répertoire courant, la commande `cp fichier1 fichier2` a un comportement différent de celui décrit plus haut ...

Application : Prédire ce que réalise la séquence suivante en sachant que la commande `touch` sert à créer un fichier vide (ou mettre à jour un fichier existant) :

```
$ cd
$ mkdir -p dossier1/dossier2
$ cd dossier1/dossier2
$ touch un_fichier
$ cp un_fichier ..
$ cd ..
$ mv un_fichier autre_fichier
$ mv dossier2 dossier3
$ mv dossier3 ..
$ mv autre_fichier ../dossier3
$ cd ..
$ rmdir dossier1
$ cd dossier3
$ cp un_fichier ton_fichier
```

I.4. Caractères génériques

```
$ touch a.out ab123 fichier1 fichier2 fichier3 fichier4
$ ls
a.out abc123 fichier1 fichier2 fichier3 fichier4
$ ls fi*
```

```
fichier1 fichier2 fichier3 fichier4
$ ls fichier[12]
fichier1 fichier2
$ ls fichier[1-3]
fichier1 fichier2 fichier3
$ ls ????3
ab123 fichier3
```

II. Droits d'accès

Se placer dans un répertoire où se trouvent des fichiers textes et des fichiers exécutables (par exemple le dossier `commandes`). Comparer leurs droits d'accès (`ls -l`).
Changer les droits d'un fichier texte, par exemple `fichier.txt` : `chmod o-r fichier.txt`
Vérifier : `ls -l`

Les droits d'accès d'un fichier sont définis par trois champs **`rwx`** respectivement associés aux droits propres à l'utilisateur, à son groupe d'utilisateurs et aux autres utilisateurs. La présence (l'absence) d'un droit peut être codée en binaire :

`r-x` est codé 101 (5 en décimal)

`-wx` est codé 011 (3 en décimal)

La commande `chmod 750 fichier.txt` change donc les droits du fichier en attribuant à chaque champ les codes respectifs 7 (`rwx`), 5 (`r-x`) et 0 (`---`).

- Enlever le droit `r` du répertoire `commandes` pour les utilisateurs du même groupe que vous.
- Vérifier qu'un utilisateur du même groupe que vous ne peut pas **lister** le contenu de votre répertoire `commandes` mais qu'il peut le **traverser** (se placer dedans).
- Remettre le droit `r` et enlever le droit `x` du répertoire `commandes` pour les utilisateurs du même groupe.
- Vérifier qu'un utilisateur du même groupe peut **lister** le contenu de votre répertoire `commandes` mais ne peut pas le **traverser**.

III. Environnement utilisateur

III.1. Variables d'environnement

L'environnement de travail de chaque utilisateur est en partie défini par un certain nombre de variables **Shell**. Toutes les variables sont de type chaîne de caractères. Exemple : `$HOME` et `$PATH` contiennent respectivement votre répertoire de travail **principal** (*home directory*) et la liste des références des répertoires où doit être recherchée toute commande à exécuter.

```
$ echo $HOME
$ echo $PATH
```

$HOME et $PATH servent entre autre lorsqu'on écrit des scripts Shell récursifs.
Remarque : ~ vaut $HOME en **C-Shell** et en **Bash Shell** (*Bourne-Again Shell*).
Une autre variable d'environnement est celle qui détermine l'écran sur lequel doivent se faire les affichages.
`$ echo $DISPLAY`

III.2. Fichiers de configuration

La configuration de l'environnement se fait grâce à un fichier de configuration situé sous le répertoire principal de chaque utilisateur ($HOME) : **`.bashrc`**.
Ce fichier de configuration référence des fichiers de configuration généraux du système, notamment `/etc/bash.bashrc`.

IV. Introduction aux processus

La commande `ps` permet d'afficher la liste des processus actifs sur la station. Ces processus cohabitent et partagent le potentiel de calcul du processeur.

```
$ ps
$ ps -u   # Processus de l'utilisateur courant
$ ps -aux # Tous les processus de tous les utilisateurs connectés
```

À partir d'un **terminal**, lancer une nouvelle fenêtre en tapant la commande "`xfce4-terminal`" ou bien "`terminator`". Taper CTRL+Z dans la fenêtre initiale puis `fg`. Comparer avec la commande "`xfce4-terminal &`" (ou "`terminator &`").

V. Redirections

Chaque commande Unix possède une entrée, une sortie et une sortie erreur standards. Par exemple la commande `ls` a pour sortie standard l'écran. Les redirections consistent à rediriger les entrées et sorties standards vers d'autres destinations (un fichier au lieu du clavier ou un fichier au lieu de l'écran).

Rediriger la sortie standard de la commande `ls` dans le fichier `fich` : `ls -l > fich`. Si `fich` n'existait pas il est créé sinon il est remplacé (*écrase*). Afficher le contenu de `fich` : `cat fich`.
Il est possible de rediriger en concaténant : `ls -l >> fich`. Dans ce cas, si le fichier existe déjà la sortie de la commande est ajoutée à la fin du fichier (*concaténée*), sinon il est créé.

La commande wc affiche par défaut le nombre de lignes, de mots et de caractères d'une suite de caractères tapée au clavier et terminée par CTRL+D. Tester la commande wc.
Ce n'est guère pratique ni utile. Il est possible de rediriger l'entrée standard (le clavier) : la commande `wc < fich` affiche à l'écran le nombre de lignes, de mots et de caractères du fichier `fich`. Cette utilisation est beaucoup plus fréquente.

L'utilisation de tubes (*pipes*) permet de récupérer la sortie d'une commande pour l'utiliser comme entrée d'une autre.

La syntaxe est `commande1 | commande2`.
Exemples :

```
$ ls | grep 'f'
$ ls -l | wc -l
$ ps ux | grep 'bash'
```

- Placez-vous dans votre répertoire `commandes` sous votre répertoire principal et exécutez le programme `moyenne` qui calcule et affiche la moyenne de deux entiers (`cat moyenne.cpp` pour consulter le code source).
- Éditez un fichier `entree.in`, par exemple avec l'éditeur `gedit`. Tapez deux entiers, sur la même ligne ou un sur chaque ligne.
- Exécutez la commande : `./moyenne < entree.in`

VI. Réseau

VI.1. `ssh` et `telnet`

La commande `ssh` sert à se connecter (en console) sur une machine distante.

```
$ ssh <ordinateur_distant> -l <login> -p <port>
$ # Ou bien :
$ ssh <login>@<ordinateur_distant> -p <port>
```

`ssh` est une abbréviation de *secure shell*, ce qui signifie que toutes les données sont **cryptées** entre l'ordinateur local et l'ordinateur distant.
Le protocole `ssh` utilise par défaut le port **22**.

La commande `telnet` permet de faire la même chose mais de manière non sécuritaire (les données transitent non cryptées) :

```
$ telnet <ordinateur_distant>
```

Le protocole `telnet` utilise par défaut le port **23**.
La commande `telnet` est souvent utilisée pour vérifier si un port est ouvert et si c'est le cas, interroger le service correspondant. Exemple :

```
$ telnet serveur.web.com 80
```

VI.2. `ftp`

La commande `ftp` (f*ile* t*ransfert* p*rotocol*) permet de lancer un processus de transfert de fichiers entre le site local et un site distant dont le nom peut être donné en argument.
Le protocole `ftp` utilise par défaut le port **21**.
Il existe aussi le protocole **`sftp`** (*secure shell ftp*) pour le transfert de fichiers sécurisé, utilisant le même port que `ssh` (22).

VI.3. Autres

- Sous certains Unix/Linux, la commande `rusers` (options possibles `-a`, `-l`) permet de savoir qui est connecté sur les ordinateurs du réseau local.
- La commande w permet de savoir qui est connecté sur l'ordinateur local et quelles tâches chacun exécute.
- La commande `whoami` affiche le nom de l'utilisateur courant.
- La commande `id` affiche le numéro de l'utilisateur courant (*uid*) ainsi que son numéro de groupe primaire (*gid*) et la liste de ses numéros de groupes secondaires (*groups*).
- La commande who permet de savoir qui est connecté sur l'ordinateur local.
- La commande `finger` permet d'afficher certaines informations sur les utilisateurs enregistrés dans le système local.
- La commande `curl` sert à transférer de l'information d'une URL (**téléchargement**). Elle est souvent utilisée pour obtenir de l'information telle que :
  - La météorologie :

```
$ curl wttr.in/:help           # Manuel en ligne
$ curl wttr.in                 # Météo locale (basée sur l'IP)
$ curl wttr.in/~Dieppe+France  # Météo à Dieppe, France
$ curl wttr.in/~Dieppe+Canada  # Météo à Dieppe, Canada
$ curl wttr.in/46.105,-64.7819 # Météo à l'UdeM (coordonnées GPS)
$ curl wttr.in/Moon            # Phases de la lune
```

  - Les informations internet côté client :

```
$ curl ipinfo.io/ip            # Adresse IP cliente (WAN)
$ curl ipinfo.io               # IP, hôte, ville, pays, etc.
$ curl ipinfo.io/207.46.13.41  # Infos sur une IP spécifique
```

  - *ASCII Art* :

```
$ # Statique :
$ curl -s http://artscene.textfiles.com/asciiart/panda
$ # Animée :
$ curl -s http://artscene.textfiles.com/vt100/wineglas.vt | \
        pv -L1000 -q
```

VII. Divers

VII.1. Archivage, compression

La commande `tar` sert à archiver une arborescence de fichiers et de répertoires.

- Créer une petite arborescence de fichiers et de répertoires à l'aide des commandes `mkdir` (création d'un répertoire) et `touch` (mise à jour/création d'un fichier).
  Exemple :

```
$ mkdir dossier1
$ mkdir dossier1/dossier2
$ # Ou bien une seule instruction : mkdir -p dossier1/dossier2
$ touch dossier1/dossier2/toto
```

- Archiver l'arborescence :

```
$ tar cvf archive.tar dossier1
```

- Effacer l'arborescence :

```
$ rm -r dossier1
```

- Restituer l'arborescence :

```
$ tar xvf archive.tar
```

- Vérifier que toute l'arborescence a bien été restaurée :

```
$ ls -lR dossier1
```

La commande `gzip` (*GNU Zip*) sert quant à elle à réduire la taille de (gros) fichiers, selon le principe du codage de Lempel-Ziv (technique de **compression sans perte**). `gzip` remplace la commande `compress` qu'on trouve encore sur les systèmes Unix traditionnels. Un fichier compressé avec `gzip` est remplacé par un fichier de même nom et d'extension `.gz`. L'opération inverse (décompression) se fait avec la commande `gzip -d`.
Sur les systèmes Unix/Linux récents, on trouve maintenant les commandes `bzip2` (compression) et `bunzip2` (décompression) qui utilisent une autre technique statistique que le codage de Lempel-Ziv, offrant une meilleure compression (**sans perte**) [1].

- Compresser le fichier `archive.tar` : `bzip2 archive.tar`
- Faire la liste des fichiers : `ls -l`
- Décompresser le fichier `archive.tar.bz2` : `bzip2 -d archive.tar.bz2`
- Vérifier que le fichier `archive.tar` a bien la même taille et les mêmes droits qu'initialement.
- **Remarque :** la commande `tar` permet de décompresser **et** désarchiver en une seule opération grâce aux options suivantes : `tar xvfj archive.tar.bz2`

VII.2. Hôtes locaux

```
$ nmap -sn 192.168.1.0/24  # hôtes actifs sur le réseau local
```

VII.3. Recherche en ligne de commande

```
$ man googler  # Manuel de la commande
$ googler "INFO4108 Programmation Unix"
```

1. http://www.bzip.org

VII.4. Commandes "détachées"

```
$ man nohup  # Manuel de la commande
$ # Exemple :
$ nohup find ~/Documents -name '*.pdf' -ls &
```

Un fichier nohup.out est créé / concaténé dans le dossier courant.

```
$ man screen           # Manuel de la commande
$ screen -S maSession  # Création d'une session
$ # CTRL-A-D pour se détacher de la session.
$ # Plus tard, de n'importe quel terminal :
$ screen -r maSession  # Reprise en main de la session ("resume")
```

VII.5. Conversions en ligne de commande

```
$ man ssconvert  # Pour les feuilles de calcul. Installer le paquet gnumeric.
$ ssconvert fichier.csv fichier.xlsx  # Conversion csv --> xlxs
$ ssconvert feuille.xls feuille.csv   # Conversion xls --> csv

$ man convert  # Pour les images. Installer le paquet imagemagick.
$ convert image.png image.jpg  # png --> jpg
$ # png --> jpg avec réduction de 50% :
$ convert image.png -resize 50%
```

VII.6. Trucs et astuces !

- Taille de dossiers : `du -<options> <noms_de_dossiers`
- Facteurs premiers d'un nombre : `factor <nombre>`
- `sl` (Pour les dyslexiques !)
- `cowsay <message>`
- `espeak <message>`
- `nl <nom_de_fichier>`
- `tree [<noms_de_dossiers>]`

VIII. Premiers scripts Shell

Écrivez un premier script Shell qui affiche `bonjour` en suivant les étapes suivantes :

```
$ gedit script.sh  # Ou bien gvim script.sh etc.
```

Tapez sur la première ligne : `#!/bin/bash`
Puis sur une autre ligne : `echo "bonjour"`

```
$ chmod +x script.sh  # Pour rendre le fichier exécutable
$ ./script.sh         # Pour exécuter depuis le répertoire courant
```

Éditez le fichier et ajoutez les lignes :

```
echo "Les 5 premiers : " $1 $2 $3 $4 $5
echo "Tous : " $*
echo "Nombre : " $#
```

Puis exécutez le script avec plusieurs paramètres :

```
$ ./script.sh Un Deux Trois Quatre Cinq Six Sept Huit Neux Dix Onze Douze
```

Taper les commandes suivantes en essayant de prédire la réponse de l'interprète Shell pour chaque ligne :

```
a=pwd   # Affectation d'une chaîne à la variable a
b=chose # Affectation d'une chaîne la variable à b

echo $a # Substitution puis echo
echo $b # Substitution puis echo
echo $c # Substitution sur une variable non définie puis echo

echo "$a" # Substitution puis echo
echo "$b" # Substitution puis echo
echo "$c" # Substitution sur une variable non définie puis echo

echo '$a' # Interdiction de substitution puis echo
echo '$b' # Interdiction de substitution puis echo
echo '$c' # Interdiction de substitution puis echo

echo $a # Substitution, évaluation puis echo
echo $b # Substitution, évaluation puis echo
echo $c # Substitution, évaluation sur une variable non définie puis echo

$a # Substitution
$b # Substitution
$c # Substitution

"$a" # Substitution
"$b" # Substitution
"$c" # Substitution

'$a' # Interdiction de substitution
'$b' # Interdiction de substitution
'$c' # Interdiction de substitution

'$a' # (backticks) Substitution puis évaluation
```

```
'$b' # (backticks) Substitution puis évaluation
'$c' # (backticks) Substitution puis évaluation
```