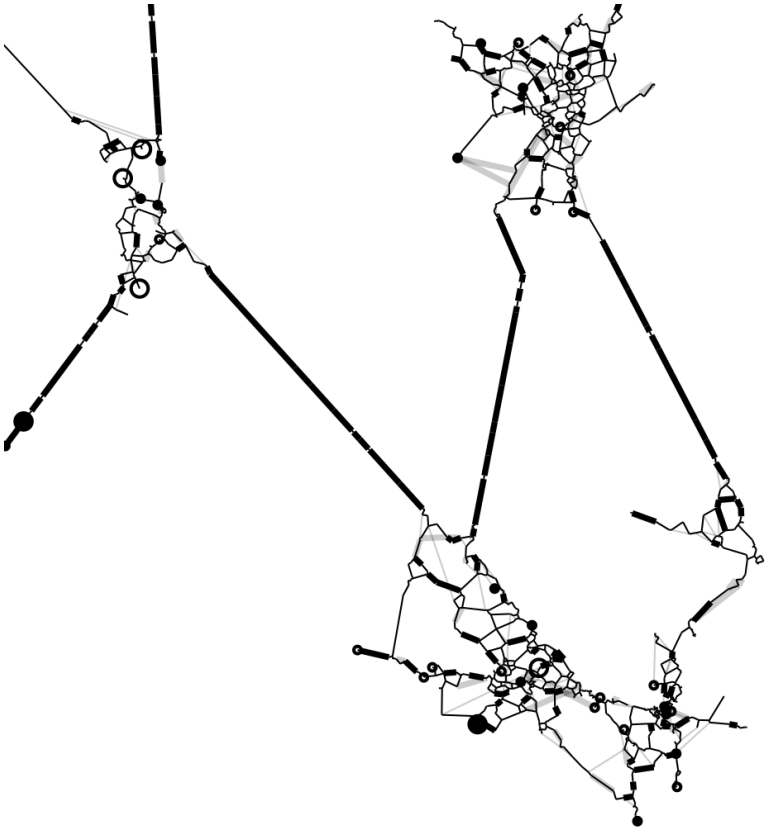
IUT de Villetaneuse
Département R&T

Module M22
2024-2025
Introduction au système Unix

Exercices de travaux pratiques



Emmanuel Viennet
emmanuel.viennet@univ-paris13.fr

Image de couverture : http://inconvergent.net/

TP N° 1 - Rappels : utilisation de base

Objectif : se connecter à un système UNIX, l'utiliser en mode console (terminal), utiliser les commandes de base pour manipuler des fichiers.
Vous rédigerez un compte rendu, sur lequel vous indiquerez la réponse à chaque question, vos explications et commentaires (interprétation du résultat), et le cas échéant la ou les commandes utilisées.

EXERCICE 1 - Connexion au système

Les exercices peuvent être réalisés sur toute machine Linux (la distribution ayant à ce niveau peu d'importance, Debian ou Ubuntu étant les plus recommandées).

Vérifiez (`echo $SHELL`) que votre shell de connexion (celui lancé par défaut) est `/bin/bash`.

Vous pouvez ouvrir une (ou plusieurs) fenêtres d'interpréteurs de commandes (shell).

EXERCICE 2 - Commandes de base

Attention, UNIX fait la différence entre les majuscules et les minuscules. La plupart des commandes doivent s'écrire en minuscules.
Séparez toujours la commande de ses arguments par un ou plusieurs espaces (par exemple, écrire `ls -l` et non pas `ls-l`).

1- Commandes de base : révisez l'utilisation des commandes `cd, mkdir, ls, rmdir, rm, cp, man, date, pwd, mv, echo`.

Pour chaque commande, décrire en une phrase ce qu'elle fait et indiquer le rôle des options indiquées entre crochets (vous devrez connaitre ces options par cœur) :

- `cd`
- `mkdir [-p]`
- `rmdir`
- `pwd`
- `man`
- `ls [-l] [-a] [-R] [-1]`
- `rm [-i] [-r]`
- `cp [-i] [-r] [-a]`
- `mv [-i]`
- `date`
- `echo [-n]`

Pour gagner du temps lors des différents TP, organisez bien vos fichiers. Une sauvegarde personnelle sur clé USB ou sur un espace dans le cloud (drive, dropbox...) pourra vous servir. Vous devez avoir dans votre répertoire de connexion les répertoires suivants :

- `bin` : scripts et commandes personnels ;
- `tmp` : essais temporaires, à effacer régulièrement.
- un répertoire par module, contenant un sous-répertoire par TP (`TP01, TP02, ...` : exercices du TP 1, 2 ...).

Évitez toujours de laisser des fichiers dans votre répertoire de connexion ou sur votre bureau. Utilisez les sous-répertoires. Cette discipline vous fera gagner en efficacité à long terme.

2- *Commande man*

On peut chercher un mot-clef interactivement lors de la visualisation du manuel d'une commande (la commande `man` utilise la commande `less` pour afficher la documentation).

La recherche est lancé en appuyant sur la touche `/` (voir le manuel de la commande `less` pour plus de détails).

- chercher dans le manuel de `less` le mot “pattern”.

3- *Commande ls*

En utilisant la commande `ls` et ses différentes options (voir `man ls`), visualiser le contenu de votre répertoire courant de la façon suivante :

a) Liste simple.

b) Liste montrant les fichiers cachés (ceux dont le nom commence par ”.”). On remarquera la présence des 2 entrées ”.” et ”..”.

c) Liste avec descriptif complet de chaque référence (droits, nombres de liens, dates, taille user group ...).

d) Liste avec descriptif complet et avec un format plus compréhensible concernant la taille des fichiers.

e) Liste récursive (descend dans les sous-répertoires).

f) Liste par ordre chronologique (la commande “touch” peut servir à changer la date de modification d'un fichier).

g) Liste par date d'accès au lieu de la date de création. Pour constater un changement, utiliser la commande cat ”nom de fichier” pour modifier la date du dernier accès.

h) Liste simple du contenu avec affichage du type de fichier (répertoire `/`, lien symbolique `@`, exécutable `*`).

i) Liste avec numero inode. (vous pouvez vérifier en créant avec la commande `ln` un lien physique vers un fichier existant).

EXERCICE 3 - *Redirections*

La *redirection* de la sortie d'une commande consiste à envoyer ce qu'elle affiche dans un fichier (l'affichage est alors supprimé). Pour rediriger une commande vers le fichier :

```
$ commande > fichier
```

Attention, le fichier indiqué est alors supprimé (écrasé, remplacé).

Si vous voulez ajouter à la fin d'un fichier existant, utiliser `>>` :

```
$ commande >> fichier
```

1- A l'aide d'une redirection et de la commande `echo`, créez un fichier contenant la ligne de texte : « bonjour ASUR ».

2- Ajouter la ligne de texte « Hello » au fichier précédemment créé.

3- Que fait la commande `wc [-l]` ?

4- A l'aide des commandes `ls` et `wc` (avec options si besoin) afficher le nombre de fichiers et sous-répertoire présents dans le répertoire `/etc`.

EXERCICE 4 - *Méta-caractères*

1- Python est un langage interprété très utilisé pour l'administration système (voir `http://www.python.org`). On le lance en mode interactif via la commande "python". La commande "python fichier.py" exécute le script python contenu dans le fichier "fichier.py".

Devinez ce que fait le programme Python suivant :

```
import sys

n = int(sys.argv[1])

for i in range(n):
    f = open( 'f' + str(i), 'w' )
    f.write( str(i) + '\n' )
    f.close()
```

Créez un fichier `genf.py` dans votre répertoire de TP contenant le programme précédent, puis lancez `python genf.py 16`

Qu'observez-vous ? Quelle est la taille en octets des fichiers créés ? Pourquoi ?

2- A l'aide d'une seule commande shell, créez un fichier "tous" dont le contenu soit la concaténation des fichiers précédement créés.

3- Quelle est la taille du fichier `tous` ? Combien de lignes comporte-t-il ?

4- A l'aide des commandes `grep` et `wc`, afficher le nombre de lignes du fichier `tous` qui contiennent le chiffre 1.

5- A l'aide des commandes `cut` et `sort`, afficher la liste des noms de login définis sur votre système, triée par ordre alphabétique (voir le fichier `/etc/passwd`).

6- Afficher les noms de tous les fichiers de `/usr/include` qui commencent par "`std`" et terminent par ".h".

EXERCICE 5 - Commande `find`

1- Afficher (avec `find`) les noms de tous les fichiers du répertoire `/usr` ayant une taille supérieure à 250Ko.

2- Afficher les noms de tous les fichiers du répertoire `/var` ayant été modifiés *après* votre répertoire de connexion.

3- A l'aide des commandes `find` et `grep`, afficher toutes les lignes contenant le mot `define` dans les fichiers d'extension `.h` situés dans le répertoire `/usr/include` et tous ses sous-répertoires.

EXERCICE 6 - Variables d'environnement en shell (*bash*)

1- Afficher la liste des variables d'environnement. Quel genre d'informations trouve-t-on ?

2- Le shell recherche les commandes dans la liste des répertoires indiqués dans la variable d'environnement `PATH`.

1. Quelle est la valeur de `PATH` ?
2. Créer (s'il n'existe pas déjà) dans votre répertoire de connexion un sous-répertoire nommé `outils` et y placer un exécutable (par exemple un script shell).
3. Ajouter ce répertoire `bin` à votre `PATH`.
4. Vérifier que vous pouvez maintenant lancer l'exécution de votre script quel que soit le répertoire courant.
5. Pour modifier le `PATH` de façon permanente, placer la commande de modification (3) dans le fichier de configuration de votre shell (`~/.bashrc`). A l'avenir, vous pouvez placer vos exécutables préférés dans votre répertoire `outils`. (en général, on utilise plutôt `~/bin` pour cela.

EXERCICE 7 - Propagation des variables d'environnement.

Étudier la séquence de commandes shell suivante :

```
 0     echo $ZORGLUB           ; cette var. n'existe pas !
 1     export TRUC=machin      ;  cree la variable TRUC
 2     TRAC=22
 3     echo $TRUC $TRAC        ; l'affiche
 4     bash                    ; lance un nouveau shell
 5     echo $TRUC              ; affiche la valeur de TRUC
 6     echo $TRAC              ; ?
 7     export TRUCBIS=hoho     ; une autre variable
 8     echo $TRUCBIS
 9     exit                    ; termine le second shell
10     echo $TRUC
11     echo $TRUCBIS           ; ??
```

Que se passe-t-il lors de la première commande (ligne 0) ? (comparer avec ce qui arrive dans d'autre slangages que vous connaissez si on utilise une variable qui n'existe pas).

La commande `bash` (ligne 4) ouvre un nouveau shell, qui hérite des variables de l'ancien. Que s'affiche-t-il à la ligne 11 ? Expliquer pourquoi.

EXERCICE 8 - Avec la commande `find`, écrire une commande qui affiche le nombre fichiers présents dans un répertoire donné, et *tous* ses sous répertoires et leurs descendants.

TP N° 2 - Droits, processus

Vous rédigerez un compte rendu, sur lequel vous indiquerez la réponse à chaque question, vos explications et commentaires (interprétation du résultat), et le cas échéant la ou les commandes utilisées.

EXERCICE 1 - Définition des utilisateurs (*à traiter comme utilisateur*)

1- Combien d'utilisateurs UNIX sont définis sur votre système dans `/etc/passwd` ? Parmi ceux-ci, combien utilisent le shell *bash* ou *sh* ?

2- Quel est le shell de l'utilisateur `root` ?

EXERCICE 2 - *Droits d'accès*

1- A l'aide de la commande `id`, vérifier votre identité et le(s) groupe(s) auquel vous appartenez.

2- En tant qu'utilisateur (non admin !), essayer de supprimer ou de modifier le fichier `/etc/passwd`. Que se passe-t-il ? Expliquer la situation à l'aide de la commande `ls -l`

3- Créer un petit fichier texte (de contenu quelconque), qui soit lisible par tout le monde, mais pas modifiable (même pas par vous).

4- Créer un répertoire nommé `secret`, dont le contenu soit visible uniquement par vous même. Les fichiers placés dans ce répertoire sont-ils lisibles par d'autres membres de votre groupe ?

5- Quel droits d'accès (`rwx`) faudrait-il utiliser sur un répertoire (*directory*) afin que les autres utilisateurs ne puissent pas lister son contenu (via la commande `ls`) mais puissent quand même lire les fichiers qui y sont placés ?

EXERCICE 3 - Un administrateur désire s'assurer chaque matin que tous les fichiers placés sous `/tmp/TPRT` sont *lisibles* uniquement par leur propriétaire et les membres du groupe, mais non modifiables et non exécutables.

1- Quel doit être le mode de ces fichiers et répertoires ?

2- Donnez une commande permettant d'afficher la liste des chemins des fichiers qui ne possèdent pas le bon mode.

EXERCICE 4 - *archive*

1- Créez une archive tar (tgz) contenant votre répertoire `HOME`.

2- Listez le contenu de cette archive, puis effacez la.

EXERCICE 5 - *Processus*

La commande `ps` affiche la liste des processus lancés sur le système. `ps` accepte de nombreuses options, les unes permettant d'indiquer l'ensemble de processus à afficher (basé sur la commande, l'utuilisateur ou d'autres caractéristiques),

- `ps -ef` liste tous les processus
- `ps -f -u user` liste les processus de l'utilisateur indiqué
- `ps auxww` liste tous les processus, format long avec la commande complète.

1- Combien de processus sont lancé sur votre système ?

2- Quels processus appartiennent votre utilisateur ?

3- Combien de processus appartiennent à `root` ?

4- Quel est le processus qui a le plus petit PID ?

5- On peut afficher des informations actualisées en temps réel avec la commande `top`. On la quitte en tapant « q ».

Quelle est la mémoire disponible sur votre système ? Quel processus occupe le plus le processeur ?

6- Décrire ce qu'affiche la commande `pstree`.

7- Lancez un processus par la commande `sleep 30 &`.

Quel est son processus pere ? Interrompez son execution (avant l'écoulement des 30 secondes) par l'intermédiaire de la commande `kill`.

8- Relancez le processus `sleep 30 &` et changez sa priorité pour quelle soit minimale (cela n'affectera pas beaucoup ce processus, qui reste inactif dans tous les cas !).

Vérifiez le changement avec la commande `ps`.

EXERCICE 6 - *Code de statut d'une commande*

Lorsqu'un processus Unix termine, il renvoie toujours un code de statut, qui est un nombre entier. C'est l'argument de l'appel système `exit(int)` ou la valeur retournée par la fonction principale (`main` en C).

La valeur 0 indique toujours un succès, tandis que les valeurs non nulles signalent un code d'erreur dont la signification dépend de la commande.

1- En shell, comment obtenir le statut de la dernière commande lancée ?

2- Comment terminer un script shell avec un code de statut déterminé ?

3- Quel code renvoie la commande ping vers une machine qui n'existe pas (nom introuvable) ?

4- Quel code renvoie la commande ping vers l'IP d'une machine éteinte (*request timeout*).

TP N° 3 - Shell bash

Vous rédigerez un compte rendu, sur lequel vous indiquerez la réponse à chaque question, vos explications et commentaires (interprétation du résultat), et le cas échéant la ou les commandes utilisées.

EXERCICE 1 - *méta-caractères*

Créez un repertoire nommé `html` et placez-vous à l'intérieur.

1- Que fait la succession de commandes bash suivante :

```
for i in 0 1 2 3 4 5 6 7 8 9
do
 for j in 0 1 2 3 4 5 6 7 8 9
 do
    touch $i$j.html
 done
done
```

Répétez l'opération pour faire la même chose mais avec l'extension `.xml` au lieu de `.html`. Combien au total avez-vous de fichiers dans votre répertoire `html` ?

2- Effacez les 200 fichiers (en une seule commande, en évitant les confirmations) et recréez-les (les 200 à la fois).

3- Listez tous les fichiers :

1. commençant par 0 ou par 1 (c'est-à-dire les 20 premiers) ;
2. dont le deuxième caractère est soit 7 soit 8 soit 9 ;
3. dont le deuxième caractère est soit 7 soit 8 soit 9 et l'extension est `xml`.

Recommencer en spécifiant un intervalle (de 7 à 9).

4- Effacez tous les fichiers `html` dont le nom contient un zéro (comme premier et/ou comme second caractère).

Suggestion : faites vos essais avec `ls` (ou `echo`) puis, une fois que vous aurez trouvé le motif correct, utilisez `rm` (toujours en évitant les confirmations).

5- Créer (avec la commande `touch`) un fichier nommé exactement `*vive l'IUT !*`, puis effacez le.

EXERCICE 2 - *Commandes avec variables*

L'option `-i` de la commande `ls` permet d'afficher le numéro d'inode de chaque fichier (l'inode est une structure de donnée associée à chaque fichier par le système).

1- Affecter à une variable `INODES` la liste des inodes (séparés par des blancs) des fichiers présents dans le répertoire courant.

Pour cela, utilisez la commande `awk` pour isoler la colonne donnant le numéro de l'inode (pour extraire le champ 5 d'un ensemble de lignes, la commande à utiliser serait : `awk '{print $5}'`).

EXERCICE 3 - *script*

Créer un script `test-fichier`, qui précisera le type du fichier passé en paramètre, ses permissions d'accès pour l'utilisateur

Exemple de résultats :

```
Le fichier /etc est un répertoire
"/etc" est accessible en lecture écriture exécution

Le fichier /etc/smb.conf est un fichier ordinaire
"/etc/smb.conf" est accessible en lecture.
```

EXERCICE 4 - Afficher le contenu d'un répertoire

Écrire un script bash `listedir.sh` permettant d'afficher le contenu d'un répertoire en séparant les fichiers et les (sous)répertoires.

Exemple d'utilisation :

```
$ ./listdir.sh
```

affichera :

```
--------------   Fichiers dans /etc/rc.d --------------------
rc
rc.local
rc.sysinit
--------------   Repertoires dans /etc/rc.d --------------------
init.d
rc0.d
rc1.d
rc2.d
rc3.d
rc4.d
rc5.d
rc6.d
```

EXERCICE 5 - Lister les utilisateurs

Écrire un script bash affichant la liste des noms de login des utilisateurs définis dans `/etc/passwd` ayant un UID supérieur à 500.

Indication : `for ligne in $(cat /etc/passwd)` permet de parcourir les lignes dudit fichier.

EXERCICE 6 - lecture au clavier

La commande bash `read` permet de lire une chaîne au clavier et de l'affecter à une variable. exemple :

```
echo -n "Entrer votre nom: "
read nom
echo "Votre nom est $nom"
```

La commande `file` affiche des informations sur le contenu d'un fichier (elle applique des règles basées sur l'examen rapide du contenu du fichier).

Les fichier de texte peuvent être affiché page par page avec la commande `more`.

1- Tester ces trois commandes ;

2- Écrire un script qui propose à l'utilisateur de visualiser page par page chaque fichier texte du répertoire spécifié en argument. Le script affichera pour chaque fichier texte (et seulement ceux là) la question "voulez vous visualiser le fichier machintruc ?". En cas de réponse positive, il lancera `more`, avant de passer à l'examen du fichier suivant.

EXERCICE 7 - itération, chaînes de caractères, expressions

1- On a un répertoire peuplé de fichiers dont les noms sont de la forme `dcp_1234.jpg` ou `DCP_1234.JPG`, ou encore `DCP_1234.jpg`, etc, où 1234 est une suite de chiffres quelconques.

Écrire un shell script qui renomme tous ces fichiers, pour obtenir `photo_1234.jpg` (toujours en minuscules). Les script prendra les noms des fichiers à traiter en argument sur la ligne de commande.

2- On peut lancer un émulateur de terminal coloré avec la commande

```
xterm -bg nom_de_couleur
```

On crée un fichier `colors.txt` contenant des noms de couleurs standards (voir par exemple `/etc/X11/rgb.txt`), un nom par ligne.

La variable spéciale de bash `$RANDOM` permet de générer un nombre entier aléatoire entre 0 et 32767 (voir `man bash`).

Écrire un script qui ouvre une fenêtre terminal avec un couleur de fond aléatoire, choisie dans la liste `colors.txt`.

Ajouter un bouton à la barre d'outils qui lance un terminal coloré via votre script. Tester.

EXERCICE 8 - *Boucles for,* `&&`*,* `||`*, if-then-else, case*

Utilisez les 200 fichiers (100 html et 100 xml) créés au cours de l'exercice 1 (si vous ne le avez plus, recréez-lez).

1- Éliminez tous les fichiers `.html` dont le nom contient un chiffre dans l'ensemble 0,1,2, et tous les fichiers `.xml` dont le nom contient un chiffre dans l'ensemble $7, 8, 9$ (en utilisant `rm` et les méta-caractères).

2- En utilisant une boucle `for`, créez un script qui renomme (avec `mv`) l'extension .html en .xml si le fichier .xml correspondant n'existe pas et, vice-versa, transforme l'extension .xml en .html si le fichier html correspondant n'existe pas.

1. Donner une solution qui utilise seulement `&&` et/ou `||` ;
2. donner une solution qui utilise if-then-else ;
3. donner une solution qui fabrique une variable `CAS` telle que `CAS=html` s'il n'y a pas un correspondant xml, telle que `CAS=xml` s'il n'y a pas un correspondant html, et telle `CAS=html-xml` si les deux existent, puis exécuter l'action correspondante par une instruction `case`.

3- En utilisant une boucle `for`, créez un script qui renvoie la concaténation de tous les fichiers (normaux) présents dans le répertoire courant.

TP N° 4 - Installation et mise à jour de logiciels Sauvegardes distantes

Pour ces exercices, vous utiliserez de préférence un système Linux Debian 12 (mais toute version Linux devrait être suffisante).

EXERCICE 1 - *Paquets Debian*

Les logiciels des systèmes Linux sont livrés sous forme de « paquets », qui sont des archives spéciales contenant les fichiers nécessaires (programmes, fichiers de données, configuration), mais aussi des scripts d'installation et des indications sur les logiciels qui doivent préalablement être installés sur le système pour que ce paquet fonctionne bien. Ces logiciels nécessaires sont nommés des *dépendances* : si un paquet dépend d'un autre paquet, cela signifie qu'il a besoin de celui-ci pour fonctionner.

Les principaux formats de paquets sont RPM (distributions Red Hat, Mageia) et `.deb` (distributions Debian, Ubuntu et leurs dérivés).

Sous Debian/Ubuntu, le système de gestion de paquettage est nommé APT, et utilisé principalement à travers les commandes `apt-get`, `apt-cache` et `dpkg`.

Les paquets sont identifiés par un nom (par exemple « libreoffice-writer ») et un numéro de version. Les mises à jour remplacent les paquets anciens par de nouvelles versions. Les mises à jour régulières (quotidiennes) sont essentielles pour garantir la sécurité du système (corrections de bugs pouvant être exploités par des pirates).

Les paquets sont publiés dans des *dépots*, publiés sur le Web (la liste des dépots et leurs adresses sont spécifiées dans `/etc/apt/sources.list`).

Voici les commandes à connaître :

- Rafraichir la liste des paquets : `apt-get update` (interroge les serveurs de la distribution pour télécharger la liste des paquets publiés et leurs numéros de versions)
- Installer un paquet : `apt-get install nom_paquet`
- Mettre à jour tous les paquets installés : `apt-get dist-upgrade`
- Supprimer un paquet (et ses fichiers de configuration) : `apt-get purge nom_paquet`
- Lister les paquets installés : `dpkg -l`
- Chercher un paquet (non installé) : `apt-cache search nom_paquet` (le nom peut être incomplet)
- Informations sur un paquet : `apt-cache show nom_paquet`
- Lister les dépendances d'un paquet : `apt-cache showpkg nom_paquet`

1- Combien de paquets sont installés sur votre système ?

2- Votre système est-il à jour ? Si non, mettez le à jour. Listez les paquets mis à jour dans votre compte rendu.

3- Quel paquet (non installé) fourni le logiciel LibreOffice Writer ?

4- Quels sont les dépendances de ce paquet ?

5- Installer ce logiciel. Quels menus de votre environnement sont-ils modifiés ?

EXERCICE 2 - *Surveillance de l'espace disque*

Rappel : les commande suivantes sont utiles pour surveiller l'espace disque :

- `df [-h] [repertoire]`
- `du -s [-h | -k | -m] fichier_ou_reper ...`

1- Quel est l'espace disponible sur le système de fichier qui contient votre répertoire de connexion ?

2- Quel espace occupe votre répertoire de connexion ? Et le répertoire `/usr` ?

3- Quel est le plus gros fichier dans `/usr/bin` ?

EXERCICE 3 - *Sauvegardes de fichiers sur une machine distante*

Dans cet exercice, nous allons apprendre à sauvegarder une copie des fichiers (et répertoires) d'une machine vers une autre, accessible via le réseau Internet. Nous utiliserons les commandes `ssh` (connexion à distance) et `rsync` (copie incrémentale de fichiers).

Exercice à effectuer sur une machine Linux sur laquelle vous pouvez être administrateur.

1- *Rappels sur les copies de fichiers :*

- en tant qu'utilisateur ordinaire (pas `root`), copier avec `cp` le fichier `/etc/passwd` dans votre répertoire de connexion.
  Les permissions, le propriétaire et les dates de modification sont-ils identiques sur le fichier original et sur la copie ?
- Même question en utilisant `cp -p`.
- Si on veut copier à l'identique tout un répertoire (et ses sous-répertoires), quel(les) option(s) de `cp` utiliser ?

⇒ lorsqu'on effectue des sauvegardes, il est important de copier à l'identique non seulement le contenu des fichiers, mais aussi les informations s'y rattachant (droits, propriétaire, dates, etc.).

2- *Rappels sur les connexions SSH :*

La commande `ssh` (*secure shell*) permet d'ouvrir une session ou de lancer une commande à distance, à travers un réseau.

- Demandez à votre voisin de créer un nouveau compte utilisateur sur sa machine et de vous fournir les identifiants (nom, mot de passe), ainsi que l'adresse IP de

sa machine. Vous utiliserez ce compte pour vous connecter sur sa machine, que l'on dénommera « *machine2* » dans la suite de cet exercice.

— Connectez-vous sur la *machine2* depuis votre machine, en utilisant la commande `ssh`. Qu'observez-vous ? Comment être sûr que le shell est bien sur la machine voisine ?

— SSH peut utiliser des clés cryptographiques pour éviter d'avoir à entrer le mot de passe à chaque ouverture de session distante. Créez-vous une clé RSA (indication : utiliser `ssh-keygen`) sans passphrase.

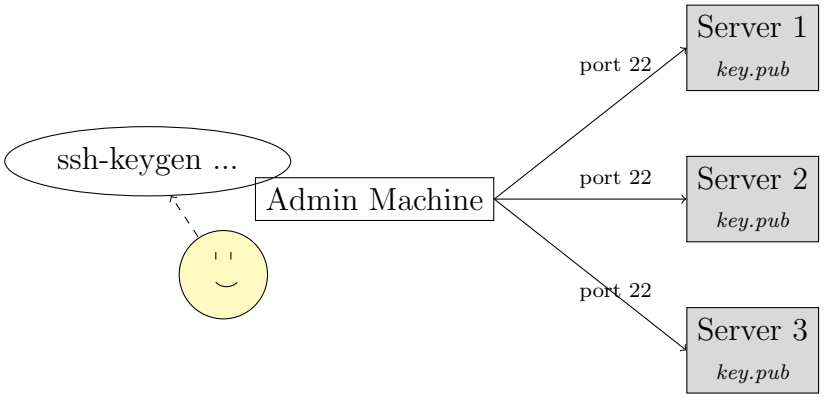
Une clé est toujours constituée d'une paire, « clé publique » (stockée dans le fichier d'extension `.pub`) et « clé privée ». La clé privée ne doit jamais être divulguée. La clé publique peut en revanche être publiée ou copiée.

Quels sont les 4 premiers et les quatre derniers octets de votre clé publique ?

— Placez votre clé publique dans le fichier `~/.ssh/authorized_keys` de votre compte sur la machine distante (*machine2*).

Vérifiez que vous pouvez maintenant vous connecter avec `ssh` sans avoir à saisir de mot de passe.

Illustration : en général, un administrateur système utilise sa clé ssh personnelle (générée sur sa machine) pour se connecter aux serveurs dont il a la charge. Pour cela, il installe sa clé publique sur ces serveurs, en la copiant dans le fichier `authorized_keys` de chacun. On peut aussi utiliser `ssh-copy-id` pour faciliter ces configurations.



3- *Copies distantes avec SSH*

On peut utiliser SSH pour copier des fichiers à distance. La commande s'appelle `scp`. Pour copier un répertoire et tout son contenu (y compris dates etc), utiliser :

```
scp -rp repertoire  utilisateur@machine:chemin
```

où `utilisateur` est le login sur la machine distante, `machine` l'adresse de celle-ci (l'IP ou le nom DNS), et `chemin` le répertoire distant dans lequel la copie sera crée.

Testez cette commande : copier l'un de vos répertoire sur la *machine2*, et vérifiez que tout est bien copié.

4- *Copies distantes incrémentales avec* `rsync`

L'inconvénient de `scp` pour les sauvegardes est que tous les fichiers sont copiés à chaque fois, ce qui peut être long si l'on manipule des répertoires volumineux (ou que le réseau est lent).

La commande `rsync` pallie à ce problème : seuls les fichiers (ou parties de fichiers) qui sont différents sur la machine source seront copiés vers la machine destination. On a donc deux répertoires, l'un sur la machine source (votre machine), l'autre sur la machine destination (ici *machine2*). Dans un premier temps, `rsync` va comparer les contenus de ces répertoires, à l'aide d'un algorithme rapide. Dans un second temps, les fichiers qui diffèrent seront copiés de la machine source vers la destination.

Il y a de nombreuses façons d'utiliser `rsync`. Voici une approche simple et efficace ; la connexion sera effectuée avec SSH, que nous avons configuré dans la question précédente.

La commande sera

```
rsync -vaze "ssh" --delete REPERTOIRE nom@machine2:DESTINATION
```

où `REPERTOIRE` est le répertoire à sauvegarder, `nom` le login sur la machine destination, et `DESTINATION` le répertoire qui contiendra la copie.

1. Créer un répertoire contenant quelques fichiers et sous-répertoires ;
2. Copier ce répertoire sur *machine2* avec la commande `rsync` précédente. Qu'observez-vous ? Vérifiez que les fichiers sont bien copiés.
3. Modifiez un fichier sur votre machine. Relancez la même commande `rsync`. Qu'observez-vous ?
4. Supprimez un fichier (`rm`) de votre machine. Relancez `rsync`. Qu'observez-vous ?

5- *Automatisation de la sauvegarde*

1. Créez un script shell qui sauvegarde l'un de vos répertoires vers l'autre machine (il suffit de placer la commande précédente dans un script).
2. Faire en sorte que ce script soit lancé automatiquement chaque minute (à l'aide de `cron`, que vous avez déjà étudié).
3. Indiquez comment vous vérifiez que cela fonctionne.
4. D'après-vous, quelles genre d'erreurs peuvent se produire durant l'exécution du script de sauvegarde ? En général, un script de ce genre n'est pas exécuté chaque minute, mais plutôt chaque nuit : que doit faire l'administrateur système pour s'assurer que les sauvegardes sont correctes ?

TP N° 5 - Shell bash

Vous rédigerez un compte rendu, sur lequel vous indiquerez la réponse à chaque question, vos explications et commentaires (interprétation du résultat), et le cas échéant la ou les commandes utilisées.

Le but de ce TP est de vous familiariser avec le langage Python, pour une utilisation simple (scripts souvent utiles en administration système). Vous utiliserez le web pour trouver des informations et documentation, et notamment sur http://:www.python.org. Pour la gestion des signaux unix utiliser le module `signal`, et pour le temps (appel sleep) le module `time`.

EXERCICE 1 - *Signaux*

1- Écrire un programme qui affiche "Hello" toutes les secondes.

2- Lancer le programme, le stopper avec CTRL-Z, le relancer, puis le tuer avec CTRL-C.

3- Modifier le programme pour qu'il ignore le CTRL-Z.

4- Modifier le programme pour qu'il affichage un message (« ouch ! »)lors de la réception de CTRL-Z.

5- Modifier le programme pour qu'il affiche "signal X recu" quel que soit le signal numéro X reçu. On peut faire une boucle en python avec la commande for, mais attention, python refuse de détourner les signaux 9 et 19 (utiliser un `try ... except`).

6- Quel signal doit-on envoyer à ce dernier programme pour le tuer ?

EXERCICE 2 - *Manipulation de fichiers*

1- Écrire un programme python nommé « compteLignes.py » qui compte le nombre de lignes dans un fichier texte (exactement comme la commande `wc -l`). Le nom du fichier sera passé sur la ligne de commande (utiliser `sys.argv`).

2- Écrire un programme python nommé « triLignes.py » qui trie les lignes d'un fichier texte (comme la commande unix `sort`).

3- AJoutez l'option `-n` à votre script de tri (qui fasse la même chose que `sort -n`.

4- A partir d'un fichier de notes formaté Prénom Nom Note1 Note2, créer un autre fichier qui contienne Nom Prénom Note1 Note2 Moyenne trié suivant l'ordre croissant des moyennes.

Les noms des fichiers seront passés sur la ligne de commande.

EXERCICE 3 - Surveillance de processus en Python

Le module `commands` de Python permet de lancer facilement des commandes (exécutables externes) et de récupérer leurs résultats Rappelons que les résultats de l'exécution d'un processus incluent :

- sa sortie standard (affichages sur stdout) ;
- sa sortie d'erreurs (affichages sur stderr) ;
- son code résultat (*exit code* ou *status*), un nombre entier, zéro signifiant que le processus s'est terminé normalement.

1- Parcourir la documentation du module `commands`. Expliquer ce que fait la fonction `getstatusoutput`.

2- A l'aide de la commande `ps auxww`, écrire un script python qui donne le nombre de processus s'étant exécutés (colonne TIME) plus d'une seconde, et affiche la liste des commandes correspondantes.

3- Écrire un script qui vérifie toutes les 5 secondes s'il existe un processus correspondant à la commande « essai ». Si ce processus existe, le signal 9 lui sera envoyé.

On utilisera la fonction `sleep` du module `time`, et le module `signal`.

Pour tester, créer un script shell nommé `essai` et affichant « hello » dix fois, avec une pause d'une seconde entre chaque affichage.

EXERCICE 4 - *Sauvegardes avec rsync* (révision)

La mise en place et la surveillance des sauvegardes des fichiers est l'une des tâches essentielles des administrateurs systèmes.

Les moyens mis en place dépendent du contexte et du niveau de sécurité attendu. Parmi les solutions fréquemment rencontrées figurent :

- utilisation de disques RAID (redondance) : gestion facile et coût modique, protection contre la panne d'un disque dur, mais pas contre la destruction (ou le vol) du serveur ;
- sauvegardes sur bandes ou disques optiques : lent, coûteux (surveillance du dispositif mécanique), de moins en moins utilisé.
- sauvegardes sur un autre serveur distant (ou dans le « cloud ») : nécessite de sécuriser le transfert réseau et l'accès à l'autre serveur (cryptage...).

Il existe des logiciels spécialisés pour la gestion des sauvegardes (un exemple est la « time machine » d'Apple Mac OS X), permettant notamment de rechercher une version précise d'un fichier (ex : mon document tel qu'il était le 31 mai 2018), et surtout de minimiser les opérations de sauvegardes (il est inutile de copier chaque jour un fichier qui n'est pas modifié).

Sous UNIX, l'un des outils (libre et gratuit) de base est la commande `rsync`. Cette commande permet de « synchroniser » le contenu de deux répertoires : l'un sera une copie conforme de l'autre (fichiers et sous-répertoires), et rsync ne copiera que les fichiers qui ont été modifié dans le répertoire original, minimisant ainsi le temps nécessaire à la copie. Le répertoire destination (la sauvegarde) peut être local (sur la même machine) ou distant (accessible en réseau, soit via un partage de fichiers comme NFS, soit via une connexion SSH).

Parmi d'autres logiciels libres intéressants pour les sauvegardes, citons : Bacula `http://www.bacula.org`, Mondo Rescue `http://mondorescue.org`, Amanda `http://www.amanda.org`.

1- Lire la documentation de la commande `rsync`. A l'aide de cette commande, créer une copie de sauvegarde du répertoire contenant votre TP. Cette copie sera située dans un autre répertoire sur votre compte utilisateur.

Attention : il est conseillé d'effectuer quelques essais sur un autre répertoire de test, pour éviter d'écraser par erreurs vos fichiers personnels ! L'option `-n` est aussi très utile pour les tests.

2- Modifier un fichier et re-synchroniser. Combien de fichiers sont transférés ? Quel est le gain par rapport à une copie complète ?

3- Si vous supprimez un fichier dans le répertoire original, est-il supprimé de la copie lors de la synchronisation ? Quelle option faut-il utiliser ?

4- Pour faire une copie sur une machine distante via SSH, quelle commande faut-il utiliser ? Comment éviter d'avoir à saisir un mot de passe à chaque synchronisation ?