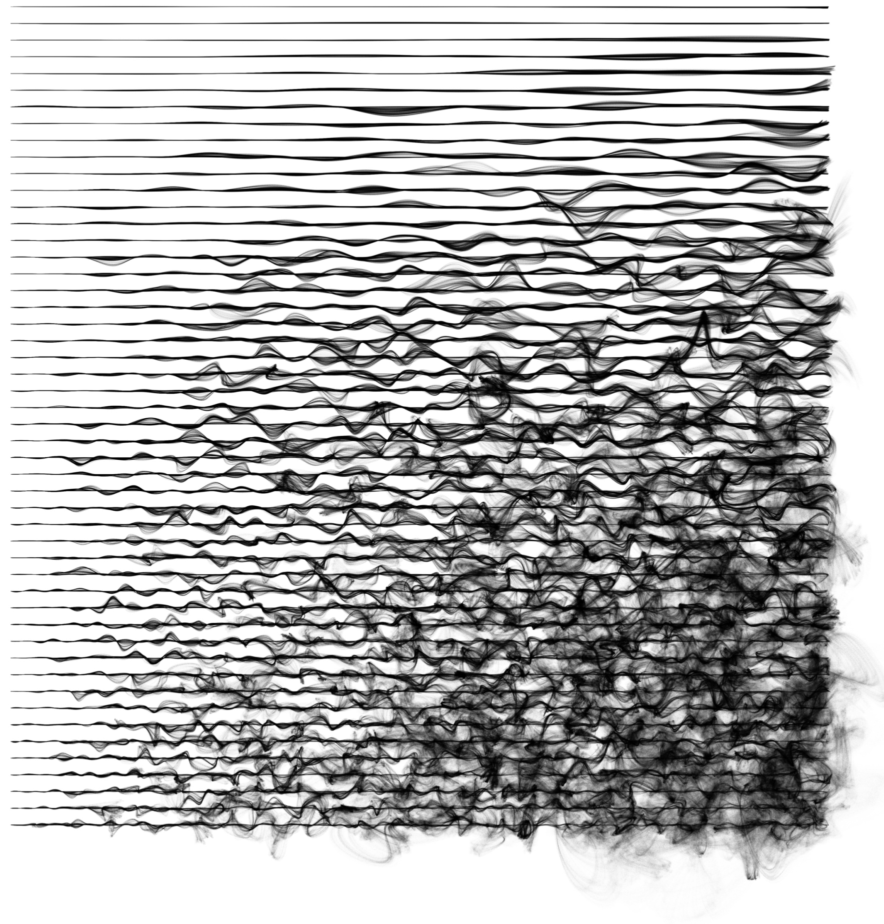
IUT de Villetaneuse
Département R&T

Module M3206
2020-2021
Automatisation des tâches d'administration



— Exercices des travaux pratiques 1 à 6 ;

— Notes de cours sur le shell Bash et les commandes de base.

Emmanuel Viennet
emmanuel.viennet@univ-paris13.fr

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

TP N° 1 - Rappels : commandes de base UNIX et Shell Bash

L'objectif de ce TP est réviser les commandes de base d'UNIX et les bases du shell Bash, qui ont été étudiées au premier semestre (module M1105 « Introduction aux systèmes informatique »). Vous pouvez au besoin vous reporter aux supports de cours de ce module (disponibles sur le Web).

Nous travaillerons avec la version Linux Debian (image nommée « bosc » en salle de TP).

Indication : vous devez chercher (avec la commande `man` *ou sur Internet) de l'aide les commandes ou fichiers à utiliser, et notez soigneusement les réponses sur votre compte rendu.*

EXERCICE 1 - Commandes de base

Attention :
- UNIX fait la différence entre les majuscules et les minuscules. Les commandes de base s'écrivent 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]`

2- Pour gagner du temps lors des différents TP, organisez bien vos fichiers. Une sauvegarde personnelle sur clé USB 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. Utilisez les sous-répertoires. Cette discipline vous fera gagner en efficacité à long terme.

3- *Commande man*

On peut chercher un mot clef interactivement lors de la visualisation du manuel d'une commande (qui se fait en réalité par l'intermédiaire du programme `less`).

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».

4- *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 :

1. Liste simple.
2. Liste montrant les fichiers cachés (ceux dont le nom commence par "."). On remarquera la présence des 2 entrées "." et "..".
3. Liste avec descriptif complet de chaque référence (droits, nombres de liens, dates, taille user group ...).
4. Liste avec descriptif complet et avec un format plus compréhensible concernant la taille des fichiers.
5. Liste récursive (descend dans les sous-répertoires).
6. Liste par ordre chronologique (la commande `touch` peut servir à changer la date de modification d'un fichier).
7. 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.
8. Liste simple du contenu avec affichage du type de fichier (répertoire `/`, lien symbolique `@`, exécutable `*`).
9. Liste avec numero inode. (vous pouvez vérifier en créant avec la commande `ln` un lien physique vers un fichier existant).

EXERCICE 2 - *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 ».

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

3- Que fait la commande `wc` ? Et `wc -l` ?

4- A l'aide des commandes `ls` et `wc` (avec options si besoin) et d'une redirection vers un fichier créé dans le répertoire `/tmp`, faire afficher le nombre de total de fichiers (ou répertoires) présents dans le répertoire `/etc`.

5- Faire la même chose en une seule ligne avec un tube (`|`).

EXERCICE 3 - Trouvez les informations suivantes et notez-les sur votre compte-rendu :

1. Quelle est l'adresse IP de votre machine ?
   Commande utilisée :
   Résultat :
2. Quelle est l'adresse IP du DNS principal ?
   Fichier ou commande utilisé(e) :
   Résultat :
3. Quelle est le nom de votre machine ?
   Fichier ou commande utilisé(e) :
   Résultat :
4. Quelle est la mémoire vive installée ?
   Fichier ou commande utilisé(e) :
   Résultat :
5. Quelle est votre répertoire de connexion (HOME) ?
   Fichier ou commande utilisé(e) :
   Résultat :
6. Quelle est l'espace disque disponible sur ce répertoire ?
   Fichier ou commande utilisé(e) :
   Résultat :

EXERCICE 4 - 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 `automatic` dans les fichiers d'extension `.h` situés dans le répertoire `/usr/include` et tous ses sous-répertoires.

EXERCICE 5 - *Méta-caractères*

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".

1- 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 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 `~/outils` 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'autres langages 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 ainsi que leurs descendants.

EXERCICE 9 - *Révision sur les tubes*

1- Quelle est la différence entre `tee` et `cat` ?

2- Que font les commandes suivantes :

```
$ ls | cat

$ ls -l | cat > liste

$ ls -l | tee liste

$ ls -l | tee liste | wc -l
```

Page blanche

TP N° 2 - 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 - Itération, chaînes de caractères, expressions

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).

EXERCICE 2 - 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 100.

Indication : `for in $(cat /etc/passwd)` permet presque de parcourir les lignes du dit fichier. Cependant, quel est le problème ? Résoudre ce problème en utilisant `cut` (avec les bons arguments) au lieu de `cat`.

EXERCICE 3 - 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 fichiers de texte peuvent être affichés page par page avec la commande `more` (ou `less`, qui est légèrement plus sophistiquée, car *less is more...*).

1- Que fait la commande `read` ? (tester et expliquer)

2- Que fait la commande `file` ? (tester et expliquer)

3- Que fait la commande `less` ?
- comment quitter `less` ?
- comment avancer d'une ligne ?
- comment avancer d'une page ?
- comment remonter d'une page ?

— comment chercher une chaîne de caractères ? Passer à l'occurence suivante ?

4- Qu'appelle-t-on un *fichier de texte* ? (ou « fichier texte »). A l'aide de la commande `file`, comment peut-on les repérer ?

5- Écrire un script qui affiche les noms de tous les fichiers de *texte* (et seulement ceux là) du répertoire spécifié en argument.

6- 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 à visualiser la question "voulez-vous visualiser le fichier machintruc ?". En cas de réponse positive, il lancera `less`, avant de passer à l'examen du fichier suivant.

EXERCICE 4 - Créer un script `test-fichier`, qui affichera en clair la nature du « fichier » passé en paramètre (fichier ordinaire ou répertoire) et ses permissions d'accès pour l'utilisateur qui lance le script.

On utilisera des conditions (if/then/else/elif), les opérateurs de test du shell (`-d`, `-f`, etc.) et les commandes de bases (`ls`, `cut`, ...).

Exemples :

```
# Exemple 1: script lancé par l'utilisateur root
$ ./test-fichier /etc
Le fichier /etc est un répertoire
"/etc" est accessible par root en lecture écriture exécution

# Exemple 2: script lancé par l'utilisateur jean
$ ./test-fichier /etc/smb.conf
Le fichier /etc/smb.conf est un fichier ordinaire qui n'est pas vide
"/etc/smb.conf" est accessible par jean en lecture.
```

EXERCICE 5 - Afficher le contenu d'un répertoire

1- Écrire un script bash `listedir.sh` permettant d'afficher le contenu d'un répertoire comme dans cet exemple :

```
$ ./listdir.sh /etc/rc.d
--------------   Contenu de /etc/rc.d --------------------
init.d
rc
rc.local
rc0.d
```

Votre script comportera nécessairement une *fonction* chargé de l'affichage du contenu du répertoire passé en argument.

2- Améliorez le script pour qu'il affiche séparément les fichiers et les (sous)répertoires. Exemple :

```
$ ./listdir.sh
--------------   Fichiers dans /etc/rc.d --------------------
rc
rc.local
```

```
--------------   Repertoires dans /etc/rc.d --------------------
init.d
rc0.d
...
```

TP N° 3 - Shell Bash (suite)

EXERCICE 1 - *Structure de contrôle (tests, if/then/else, case, while)*

1- Créer un script qui demande à l'utilisateur de saisir une note et qui affiche un message en fonction de cette note :

- "très bien" si la note est entre 16 et 20 ;
- "bien" lorsqu'elle est entre 14 et 16 ;
- "assez bien" si la note est entre 12 et 14 ;
- "moyen" si la note est entre 10 et 12 ;
- "insuffisant" si la note est inférieur à 10.

2- Modifier le script précédent pour que la note soit entrée sur la ligne de commande, et non lue au clavier.

3- Écrire un script shell qui affichera 5,4,3,2,1 en utilisant une boucle while.

4- Écrire un script qui, en utilisant la commande ifconfig, affiche la liste des interfaces réseaux configurées sur votre système. Exemple :

```
$ ./listinterfaces.sh
lo0 eth1
```

5- Ajouter l'affichage de l'adresse MAC à coté des interfaces Ethernet.

6- Écrire un script vous permettant de réaliser à l'écran le dessin suivant :

```
$ ./dessin.sh
Entrer un nombre entre 5 et 9: 9
         .
        . .
       . . .
      . . . .
     . . . . .
    . . . . . .
   . . . . . . .
  . . . . . . . .
 . . . . . . . . .
 . . . . . . . . .
  . . . . . . . .
   . . . . . . .
    . . . . . .
     . . . . .
      . . . .
       . . .
        . .
         .
```

Page blanche

TP N° 4 - Shell et processus

EXERCICE 1 - *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'utilisateur 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`.

EXERCICE 2 - *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, les valeurs non nulles 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*).

EXERCICE 3 - *Démarrage et paramétrage des services*

1- Créez un script `test-reseau.sh` qui teste périodiquement (toutes les 60 secondes, par exemple) et indéfiniment la connexion au réseau en envoyant 2 requêtes ping à la machine `www.iutv.univ-paris13.fr`. Après chaque test, le script devra ajouter à la fin du fichier `/var/log/test-reseau.log` un message donnant la date et l'heure du test ainsi que le résultat du test. Par exemple :

```
samedi 10 sept 2016, 10:19:32 (UTC+0200): test ok (2/2)
samedi 10 sept 2016, 10:20:32 (UTC+0200): test ok (2/2)
samedi 10 sept 2016, 10:21:32 (UTC+0200): test pas ok (1/2)
```

2- Créez un service `test-reseau` qui permette de lancer/arrêter ce script et de tester s'il est démarré (et, si c'est le cas, d'afficher son identifiant de processus, PID). Par exemple :

```
$ /etc/init.d/test-reseau status
Le service test-reseau est arrete.
$ /etc/init.d/test-reseau start
Service test-reseau demarre.
$ /etc/init.d/test-reseau status
Le service test-reseau est demarre (pid = 5483).
$ /etc/init.d/test-reseau stop
Service test-reseau arrete.
```

Remarque : le PID du service devra être sauvegardé dans le fichier `/var/run/test-reseau.pid`. Utilisez pour cela la variable `$!` qui contient le PID du processus en arrière-plan le plus récent.

3- Le système Debian utilise un mécanisme nommé "SysVinit" (remplacé par systemd dans les nouvelles versions) pour gérer les services, c'est à dire les processus permanents lancés au démarrage de la machine.

A l'aide de la commande `ln`, faites en sorte que le service `test-reseau` soit démarré automatiquement lorsque votre machine démarre.

Redémarrez la machine et consultez les fichiers `/var/log/boot.log` et `/var/log/test-reseau.log` pour vérifier que le service `test-reseau` a bien été démarré. Vérifiez aussi que le processus est bien présent à l'aide de la commande `ps`.

TP N° 5 - Scripts Python

EXERCICE 1 - *Introduction à Python*

Références utiles :

— http://python.developpez.com/tutoriels/cours-python-uni-paris7/
— Le site officiel, avec la doc : https://www.python.org/

1- Lire les sections I à III du cours http://python.developpez.com/tutoriels/cours-python-uni-paris7/

2- Lancer l'interpréteur et effectuer les calculs suivants, et interpréter/expliquer chaque résultat.

```
24*7
(3+2) * -4
2**8
2**1000 - 1
'IUT' + ' de Villetaneuse'
3 * 'ALLO '
```

3- Saisir le code suivant dans un fichier `essai.py`, l'enregistrer et lancer son exécution :

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

print "Bonjour !"
```

.

4- Lire les sections IV et V du cours.

Que fait le code suivant ?

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

i = 0
for x in [ 'bonjour', 'les', 'enfants' ]:
    i = i + 1
    print str(i) + ' ' + x
```

5- Des fonctions supplémentaires sont accessibles via des modules qu'il faut « importer » :

```
import math
```

```
print math.sqrt(10)
```

Le module `os` définit de nombreuses fonctions utiles pour manipuler des fichiers et accéder au système d'exploitation. Par exemple, `os.listdir( rep )` rend une liste des noms de fichiers et répertoires présents dans le répertoire indiqué (comme le fait la commande UNIX `ls`).

A l'aide de `os.listdir`, écrire un script qui affiche la liste des fichiers dans le répertoire `/tmp`.

6- Comment accéder à un argument de la ligne de commande dans un script Python ? (autrement dit, quel est l'équivalent de `$1, $2, ...`)

7- Modifiez votre script pour qu'il affiche les fichiers situés dans le répertoire indiqué sur la ligne de commande.

8- Que fait la fonction `os.system()` ?

9- Que fait la fonction `os.popen()` ? En quoi `os.popen('ls')` est-il différent de `os.system('ls')` ?

EXERCICE 2 - *Tri*

Les fonctions `open(filename)` ou `os.popen(command)` ramènent un objet « fichier ».

La méthode `readlines()` sur les objets fichiers permet de lire en une fois toutes les lignes du fichier (texte). Elle rend une liste de chaîne de caractères.

Écrire un script Python (`tri.py`) qui affiche le contenu d'un fichier (dont le nom sera spécifié sur la ligne de commande) avec les lignes triées par ordre alphanumérique. Vous réaliserez deux versions différentes du script :

1- la première utilisant `os.popen()` et la commande Unix `sort` ;

2- la seconde version utilisant la méthode Python `sort()` sur une liste de chaînes de caractères.

EXERCICE 3 -

Soit un fichier de données au format "CSV", les colonnes étant séparées par des caractères ';'. Les données sont normalement des nombres réels, mais peuvent éventuellement être manquantes (chaînes vides dues à une absence de mesure par exemple), voire du texte quelconque (annotations) qui sera ignoré.

La première ligne du fichier CSV donne le titre de chaque colonne

On veut un programme qui affiche quelques statistiques sur chaque colonne : nombre de lignes, nombre de valeurs manquantes, valeur minimale, maximale et moyenne, nombre de valeurs distinctes.

Exemple de fichier :

```
Temperature;Humidite;Pesanteur;Note
27;88;;Tropical
12;72;1.01;Tempere
14;40;0.999;
;67;1.0;thermometre HS
25;69;1.0;ideal ?
19;60;1.0;
```

Le programme affichera :

```
*** Informations sur la colonne Temperature
Nombre de lignes: 6
Nombre de valeurs manquantes: 1
Min=12.000000, Max=27.000000, Moyenne=19.4
Nombre de valeurs distinctes: 5

*** Informations sur la colonne Humidite
Nombre de lignes: 6
Nombre de valeurs manquantes: 0
Min=40.000000, Max=88.000000, Moyenne=66.0
Nombre de valeurs distinctes: 6

*** Informations sur la colonne Pesanteur
Nombre de lignes: 6
Nombre de valeurs manquantes: 1
Min=0.999000, Max=1.010000, Moyenne=1.0018
Nombre de valeurs distinctes: 3

*** Informations sur la colonne Note
Nombre de lignes: 6
Nombre de valeurs manquantes: 6
statistiques non disponibles
Nombre de valeurs distinctes: 0
```

Page blanche

TP N° 6 - Installation et mise à jour de logiciels Sauvegardes distantes

Pour ces exercices, vous utiliserez l'image Linux « Bosc » (Debian).

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-ils 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 commandes 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 (par exemple l'image Debian « bosc »).

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 ?

⇒ À retenir : 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.

1. 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.

2. 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 ?
3. 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 DSA (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 ?
4. 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.

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 répertoires 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 ?

Chapitre 4

Scripts Shell en bash

4.1 Pourquoi utiliser bash ?

Bash est une version évoluée du shell sh (le "Bourne shell"). Le shell peut être utilisé comme un simple interpréteur de commande, mais il est aussi possible de l'utiliser comme langage de programmation interprété (scripts).

La connaissance du shell est indispensable au travail de l'administrateur unix :

- le travail en "ligne de commande" est souvent beaucoup plus efficace qu'à travers une interface graphique ;
- dans de nombreux contextes (serveurs, systèmes embarqués, liaisons distantes lentes) on ne dispose pas d'interface graphique ;
- le shell permet l'automatisation aisée des tâches répétitives (scripts) ;
- de très nombreuses parties du système UNIX sont écrites en shell, il faut être capable de les lire pour comprendre et éventuellement modifier leur fonctionnement.

4.1.1 Autres versions de shell

Il existe plusieurs versions de shell : sh (ancêtre de bash), csh (C shell), ksh (Korn shell), zsh, etc. Nous avons choisi d'enseigner bash car il s'agit d'un logiciel libre, utilisé sur toutes les distributions récentes de Linux et de nombreuses autres variantes d'UNIX. Connaissant bash, l'apprentissage d'un autre shell sur le terrain ne devrait pas poser de difficultés

4.1.2 Shell ou Python ?

Nous avons vu qu'il était possible d'écrire des programmes en shell. Pour de nombreuses tâches simples, c'est effectivement très commode. Néanmoins,

le langage shell est forcément assez limité ; pour des programmes plus ambitieux il est recommandé d'utiliser des langages plus évolués comme Python ou Perl, voire des langages compilés (C, C++) si l'on désire optimiser au maximum les performances (au prix de coûts de développement plus importants).

4.1.3 Les mauvais côtés des shell

Le shell possède quelques inconvénients :
- documentation difficile d'accès pour le débutant (la page de manuel "man bash" est très longue et technique) ;
- messages d'erreurs parfois difficiles à exploiter, ce qui rend la mise au point des scripts fastidieuse ;
- syntaxe cohérente, mais ardue (on privilégie la concision sur la clarté) ;
- relative lenteur (langage interprété sans pré-compilation).

Ces mauvais côtés sont compensés par la facilité de mise en œuvre (pas besoin d'installer un autre langage sur votre système).

4.2 Variables et évaluation

Les variables sont stockées comme des chaînes de caractères.

Les variables *d'environnement* sont des variables exportées aux processus (programmes) lancés par le shell. Les variables d'environnement sont gérées par UNIX, elles sont donc accessibles dans tous les langages de programmation (voir plus loin).

Pour définir une variable :

```
$ var='ceci est une variable'
```

Attention : pas d'espaces autour du signe égal. Les quotes (apostrophes) sont nécessaires si la valeur contient des espaces. C'est une bonne habitude de toujours entourer les chaînes de caractères de quotes.

Pour utiliser une variable, on fait précéder son nom du caractère "$" :

```
$ echo $var
```

ou encore :

```
$ echo 'bonjour, ' $var
```

Dans certains cas, on doit entourer le nom de la variable par des accolades :

```
$ X=22
$ echo Le prix est ${X}0 euros
```

affiche "Le prix est de 220 euros" (sans accolades autour de X, le shell ne pourrait pas savoir que l'on désigne la variable X et non la variable X0).

Lorsqu'on utilise une variable qui n'existe pas, le bash renvoie une chaîne de caractère vide, et n'affiche pas de message d'erreur (contrairement à la plupart des langages de programmation, y compris csh) :

```
$ echo Voici${UNE_DROLE_DE_VARIABLE}!
```

affiche "Voici !".

Pour indiquer qu'une variable doit être exportée dans l'environnement (pour la passer aux commandes lancée depuis ce shell), on utilise la commande `export` :

```
$ export X
```

ou directement :

```
$ export X=22
```

4.2.1 Accès aux variables d'environnement dans des programmes

- **En langage Python** : on accède aux variables via un *dictionnaire* défini dans le module `os` : `os.environ`.
  Ainsi, la valeur de la variable `$PATH` est notée `os.environ['PATH']`.
- **En langage C** : les fonctions `getenv` et `setenv` permettent de manipuler les variables d'environnement. Le programme suivant affiche la valeur de la variable `PATH` :

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
    char *var = getenv("PATH");
    printf("la valeur de PATH est %s\n", var );
}
```

4.2.2 Évaluation, guillemets et quotes

Avant évaluation (interprétation) d'un texte, le shell substitue les valeurs des variables. On peut utiliser les guillemets (`"`) et les quotes (`'`) pour modifier l'évaluation.

- les guillemets permettent de grouper des mots, sans supprimer le remplacement des variables. Par exemple, la commande suivante ne fonctionne pas :
```
$ x=Hauru no
bash: no: command not found
```
  Avec des guillemets, c'est bon :
```
$ x="Hauru no"
```
  On peut utiliser une variable entre guillemets :
```
$ y="Titre: $x Ugoku Shiro"
$ echo $y
Titre: Hauru no Ugoku Shiro
```
- les quotes (apostrophes) groupent les mots et suppriment toute évaluation :
```
$ z='Titre: $x Ugoku Shiro'
$ echo $z
Titre: $x Ugoku Shiro
```

4.2.3 Expressions arithmétiques

Normalement, bash traite les valeurs des variables comme des chaînes de caractères. On peut effectuer des calculs sur des nombres entiers, en utilisant la syntaxe `$(( ... ))` pour délimiter les expressions arithmétiques :

```
$ n=1
$ echo $(( n + 1 ))
2
$ p=$((n * 5 / 2 ))
$ echo $p
2
```

4.2.4 Découpage des chemins

Les scripts shell manipulent souvent chemins (*pathnames*) et noms de fichiers. Les commandes `basename` et `dirname` sont très commodes pour découper un chemin en deux partie (répertoires, nom de fichier) :

```
$ dirname /un/long/chemin/vers/toto.txt
/un/long/chemin/vers
$ basename /un/long/chemin/vers/toto.txt
toto.txt
```

4.2.5 Évaluation de commandes

Il est courant de stocker le résultat d'une commande dans une variable. Nous entendons ici par "résultat" la *chaîne affichée par la commande*, et non son code de retour.

Bash utilise plusieurs notations pour cela : les *back quotes* (`) ou les parenthèses :

```
$ REP=`dirname /un/long/chemin/vers/toto.txt`
$ echo $REP
/un/long/chemin/vers
```

ou, de manière équivalente :

```
$ REP=$(dirname /un/long/chemin/vers/toto.txt)
$ echo $REP
/un/long/chemin/vers
```

(attention encore une fois, pas d'espaces autour du signe égal).

La commande peut être compliquée, par exemple avec un tube :

```
$ Fichiers=$(ls /usr/include/*.h | grep std)
$ echo $Fichiers
/usr/include/stdint.h /usr/include/stdio_ext.h
/usr/include/stdio.h /usr/include/stdlib.h
/usr/include/unistd.h
```

4.2.6 Découpage de chaînes

Bash possède de nombreuses fonctionnalités pour découper des chaînes de caractères. L'une des plus pratiques est basée sur des motifs.

La notation `##` permet d'éliminer la *plus longue* chaîne en correspondance avec le motif :

```
$ Var='tonari no totoro'
$ echo ${Var##*to}
ro
```

ici le motif est `*to`, et la plus longue correspondance "tonari no toto"[1]. Cette forme est utile pour récupérer l'extension (suffixe) d'un nom de fichier :

1. Pour ceux qui ne maîtrisent pas le japonais, je crois que $Var signifie "mon voisin Totoro".

```
$ F='rep/bidule.tgz'
$ echo ${F##*.}
tgz
```

La notation `#` (un seul `#`) est similaire mais élimine la *plus courte* chaîne en correspondance :

```
$ Var='tonari no totoro'
$ echo ${Var#*to}
nari no totoro
```

De façon similaire, on peut éliminer la fin d'une chaîne :

```
$ Var='tonari no totoro'
$ echo ${Var%no*}
tonari
```

Ce qui permet de supprimer l'extension d'un nom de fichier :

```
$ F='rep/bidule.tgz'
$ echo ${F%.*}
rep/bidule
```

`%` prend la plus courte correspondance, et `%%` prend la plus longue :

```
$ Y='archive.tar.gz'
$ echo ${Y%.*}
archive.tar
$ echo ${Y%%.*}
archive
```

4.3 Exécution conditionnelle

L'instruction `if` permet d'exécuter des instructions si une condition est vraie. Sa syntaxe est la suivante :

```
if [ condition ]
then
    action
fi
```

`action` est une suite de commandes quelconques. L'indentation n'est pas obligatoire mais très fortement recommandée pour la lisibilité du code. On peut aussi utiliser la forme complète :

```
if [ condition ]
then
    action1
else
    action2
fi
```

ou encore enchaîner plusieurs conditions :

```
if [ condition1 ]
then
    action1
elif [ condition2 ]
then
    action2
elif [ condition3 ]
then
    action3
else
    action4
fi
```

Opérateurs de comparaison

Le shell étant souvent utilisé pour manipuler des fichiers, il offre plusieurs opérateurs permettant de vérifier diverses conditions sur ceux-ci : existence, dates, droits. D'autres opérateurs permettent de tester des valeurs, chaînes ou numériques. Le tableau ci-dessous donne un aperçu des principaux opérateurs :

Opérateur	Description	Exemple
	Opérateurs sur des fichiers	
-e *filename*	vrai si *filename* existe	`[ -e /etc/shadow ]`
-d *filename*	vrai si *filename* est un répertoire	`[ -d /tmp/trash ]`
-f *filename*	vrai si *filename* est un fichier ordinaire	`[ -f /tmp/glop ]`
-L *filename*	vrai si *filename* est un lien symbolique	`[ -L /home ]`
-r *filename*	vrai si *filename* est lisible (r)	`[ -r /boot/vmlinuz ]`
-w *filename*	vrai si *filename* est modifiable (w)	`[ -w /var/log ]`
-x *filename*	vrai si *filename* est exécutable (x)	`[ -x /sbin/halt ]`
file1 -nt *file2*	vrai si *file1* plus récent que *file2*	`[ /tmp/foo -nt /tmp/bar`
file1 -ot *file2*	vrai si *file1* plus ancien que *file2*	`[ /tmp/foo -ot /tmp/bar`
	Opérateurs sur les chaînes	
-z *chaine*	vrai si la *chaine* est vide	`[ -z "$VAR" ]`
-n *chaine*	vrai si la *chaine* est non vide	`[ -n "$VAR" ]`
chaine1 = *chaine2*	vrai si les deux chaînes sont égales	`[ "$VAR" = "totoro" ]`
chaine1 != *chaine2*	vrai si les deux chaînes sont différentes	`[ "$VAR" != "tonari" ]`
	Opérateurs de comparaison numérique	
num1 -eq *num2*	égalité	`[ $nombre -eq 27 ]`
num1 -ne *num2*	inégalité	`[ $nombre -ne 27 ]`
num1 -lt *num2*	inférieur (<)	`[ $nombre -lt 27 ]`
num1 -le *num2*	inférieur ou égal (<=)	`[ $nombre -le 27 ]`
num1 -gt *num2*	supérieur (>)	`[ $nombre -gt 27 ]`
num1 -ge *num2*	supérieur ou égal (>=)	`[ $nombre -ge 27 ]`

Quelques points délicats doivent être soulignés :

– Toutes les variables sont de type chaîne de caractères. La valeur est juste convertie en nombre pour les opérateurs de conversion numérique.
– Il est nécessaire d'entourer les variables de guillemets (`"`) dans les comparaisons. Le code suivant affiche "OK" si `$var` est égale à "tonari no totoro" :

```
if [ "$myvar" = "tonari no totoro" ]
then
   echo "OK"
fi
```

Par contre, si on écrit la comparaison comme
`if [ $myvar = "tonari no totoro" ]`
le shell déclenche une erreur si `$myvar` contient plusieurs mots. En effet, la substitution des variables a lieu avant l'interprétation de la condition.

4.3.1 Scripts shell

Un script bash est un simple fichier texte exécutable (droit `x`) commençant par les caractères `#!/bin/bash` (doivent être les premiers caractères du fichier).

Voici un exemple de script :

```
#!/bin/bash

if [ "${1##*.}" = "tar" ]
then
        echo $1 est une archive tar
else
        echo $1 n\'est pas une archive tar
fi
```

Ce script utilise la variable `$1`, qui est le premier argument passé sur la ligne de commande.

Arguments de la ligne de commande

Lorsqu'on entre une commande dans un shell, ce dernier sépare le nom de la commande (fichier exécutable ou commande interne au shell) des arguments (tout ce qui suit le nom de la commande, séparés par un ou plusieurs espaces). Les programmes peuvent utiliser les arguments (options, noms de fichiers à traiter, etc).

En bash, les arguments de la ligne de commande sont stockés dans des variables spéciales :

`$0, $1, ...`	les arguments
`$#`	le nombre d'arguments
`$*`	tous les arguments

Le programme suivant illustre l'utilisation de ces variables :

```
#!/bin/bash

echo 'programme  :' $0
echo 'argument 1 :' $1
echo 'argument 2 :' $2
echo 'argument 3 :' $3
echo 'argument 4 :' $4
echo "nombre d'arguments :" $#
echo "tous:" $*
```

Exemple d'utilisation, si le script s'appelle “`myargs.sh`” :

```
$ ./myargs.sh un deux trois
programme  : ./myargs.sh
argument 1 : un
argument 2 : deux
argument 3 : trois
argument 4 :
nombre d'arguments : 3
tous: un deux trois
```

4.4 Autres structures de contrôle

Nous avons déjà évoqué l'instruction `if` et les conditions. On utilise souvent des répétitions (for) et des choix multiples (case).

4.4.1 Boucle for

Comme dans d'autre langages (par exemple python), la boucle for permet d'exécuter une suite d'instructions avec une variable parcourant une suite de valeurs. Exemple :

```
for x in un deux trois quatre
do
    echo x= $x
done
```

affichera :

```
x= un
x= deux
x= trois
x= quatre
```

On utilise fréquemment for pour énumérer des noms de fichiers, comme dans cet exemple :

```
for fichier in /etc/rc*
do
   if [ -d "$fichier" ]
   then
       echo "$fichier (repertoire)"
   else
```

```
      echo "$fichier"
   fi
done
```

Ou encore, pour traiter les arguments passés sur la ligne de commande :

```
#!/bin/bash

for arg in $*
do
  echo $arg
done
```

4.4.2 Instruction case

L'instruction case permet de choisir une suite d'instruction suivant la valeur d'une expression :

```
case "$x" in
  go)
      echo "demarrage"
      ;;
  stop)
      echo "arret"
      ;;
   *)
      echo "valeur invalide de x ($x)''
esac
```

Noter les deux ; pour signaler la fin de chaque séquence d'instructions.

4.5 Définition de fonctions

Il est souvent utile de définir des fonctions. La syntaxe est simple :

```
mafonction() {
   echo "appel de mafonction..."
}

mafonction
mafonction
```

qui donne :

```
appel de mafonction...
appel de mafonction...
```

Voici pour terminer un exemple de fonction plus intéressant :

```
tarview() {
    echo -n "Affichage du contenu de l'archive $1 "
    case "${1##*.}" in
        tar)
           echo "(tar compresse)"
           tar tvf $1
        ;;
        tgz)
           echo "(tar compresse gzip)"
           tar tzvf $1
        ;;
        bz2)
           echo "(tar compresse bzip2)"
           cat $1 | bzip2 -d | tar tvf -
        ;;
        *)
           echo "Erreur, ce n'est pas une archive"
        ;;
    esac
}
```

Plusieurs points sont à noter :

- `echo -n` permet d'éviter le passage à la ligne ;
- La fonction s'appelle avec un argument (`$1`)
  `tarview toto.tar`

Chapitre 5

Utilitaires UNIX

Cette partie décrit quelques commandes utilitaires UNIX très pratiques. Ces programmes sont livrés en standard avec toutes les versions d'UNIX.

Nous commençons par décrire l'utilisation de l'éditeur standard `vi`. Si vous avez accès à un autre éditeur de texte, vous pouvez passez cette section.

5.1 L'éditeur vi

`vi` est un éditeur de texte "plein écran" (par opposition aux éditeurs "ligne" qui ne permettaient que l'édition d'une ligne à la fois).

Comparé aux éditeurs modernes, `vi` est d'abord malcommode, mais il est assez puissant et toujours présent sur les systèmes UNIX [1]. Il est donc très utile d'en connaître le maniement de base. Nous ne décrivons ici que les commandes et modes les plus utilisés, il en existe beaucoup d'autres.

A un instant donné, `vi` est soit en mode **commande**, soit en mode **insertion** :

- **mode commande :** les caractères tapés sont interprétés comme des commandes d'édition. `vi` démarre dans ce mode, il faut donc lui indiquer (commande '`i`') que l'on veut `insérer` du texte ;
- **mode insertion :** les caractères sont insérés dans le texte édité. On peut quitter ce mode en pressant la touche "ESC" (ou "Echap" sur certains claviers).

1. Notons qu'il existe sous UNIX des éditeurs pour tous les goûts, depuis Emacs pour les développeurs jusqu'à gedit et autres jeditpour les utilisateurs allergiques au clavier (mais pas à la souris).

Appel de vi depuis le shell :

`vi` *fichier*	édite *fichier*.
`vi` +n *fichier*	commence à la ligne *n*.
`vi` -r *fichier*	récupération du fichier après un crash.

Mouvements du curseur (en mode **commande** seulement) :

Touche	**Action**
flèches	Déplace curseur (pas toujours bien configuré).
ESPACE	Avance à droite.
h	Recule à gauche.
CTRL-n	Descend d'une ligne.
CTRL-p	Monte d'une ligne.
CTRL-b	Monte d'une page.
CTRL-f	Descend d'une page.
*n*G	Va à la ligne *n* (*n* est un nombre).

Commandes passant en mode **insertion** :

Touche	**Commence l'insertion**
i	Avant le curseur.
I	Au début de la ligne.
A	A la fin de la ligne.

Autres commandes :

r	Remplace le caractère sous le curseur.
x	Supprime un caractère.
d$	Efface jusqu'à la fin de la ligne.
dd	Efface la ligne courante.
/*chaîne*	Cherche la prochaine occurrence de la chaîne.
?*chaîne*	Cherche la précédente occurrence de la chaîne.

Quitter, sauvegarder : (terminer la commande par la touche "Entrée")

:w	Écrit le fichier.
:x	Écrit le fichier puis quitte `vi`.
:q!	Quitte `vi` sans sauvegarder les changements.
!!*commande*	Exécute *commande* shell sans quitter l'éditeur.

5.2 Commandes diverses

compress [*fichier*]
: Compresse le fichier (pour gagner de l'espace disque ou accélérer une transmission réseau). Le fichier est remplacé par sa version compressée, avec l'extension '.Z'. Décompression avec `uncompress` ou `zcat`.

date
: Affiche la date et l'heure. Comporte de nombreuses options pour indiquer le format d'affichage.

diff *fichier1 fichier2*
: Compare ligne à ligne des deux fichiers texte *fichier1* et *fichier2*, et décrit les transformations à appliquer pour passer du premier au second. Diverses options modifient le traitement des blancs, majuscules etc.

  `diff` peut aussi générer un script pour l'éditeur `ed` permettant de passer de *fichier1* à *fichier2* (utile pour fabriquer des programmes de mise à jour ("*patchs*")).

file *fichier*
: Essaie de déterminer le type du fichier (exécutable, texte, image, son,...) et l'affiche.

find [options]
: Cette commande permet de retrouver dans un répertoire ou une hiérarchie de répertoires les fichiers possédant certaines caractéristiques (nom, droits, date etc..) ou satisfaisant une expression booléenne donnée.

  find parcourt récursivement une hiérarchie de fichiers. Pour chaque fichier rencontré, find teste successivement les prédicats spécifiés par la liste d'options, jusqu'au premier qui échoue ou jusqu'à la fin de la liste.

  Principales options :

  -name nom le fichier à ce nom ;

  -print écrit le nom du fichier (réussit toujours) ;

  -exec exécute une commande. {} est le fichier courant. Terminer par ; .

  -type (d : catalogue, f : fichier ordinaire, p : pipe, l : lien symbolique).

  -newer fichier compare les dates de modification ;

  -o ou ;

  -prune si le fichier courant est un catalogue, élague l'arbre à ce point.

Exemples :

```
#  Recherche tous les fichier nommes "essai"
#  a partir de la racine
find / -name essai -print

#  Recherche tous les fichier commencant par "ess"
#  a partir du repertoire courant
find . -name 'ess*' -print

#  Affiche a l'ecran le contenu de tous les fichiers .c
find . -name '*.c' -print -exec cat '{}' \;
```

(l'écriture de ce dernier exemple est compliquée par le fait que le shell traite spécialement les caractères `*`, `{`, et `;`, que l'on doit donc entourer de quotes `'`.)

head [-n] [*fichier*]
: Affiche les n premières lignes du fichier. Si aucun fichier n'est spécifié, lit sur l'entrée standard.

lpr [-P*nom-imprimante*] [*fichier*]
: Demande l'impression du fichier (le place dans une file d'attente). Si aucun fichier n'est spécifié, lit sur l'entrée standard.

  Voir aussi la commande `lp`.

  L'impression d'un fichier sous Unix passe par un spooler d'impression. Ce spooler est réalisé par un démon (c'est à dire un processus système qui s'exécute en tâche de fond).

lpq [-P*nom-imprimante*]
: Permet de connaître l'état de la file d'attente associée à l'imprimante.

lprm [-P*nom-imprimante*] *numjob*
: Retire un fichier en attente d'impression. On doit spécifier le numéro du job, obtenu grâce à la commande `lpq`.

lp [-d*nom-imprimante*] *fichier*
: Comme `lpr`, sur certains systèmes.

md5sum [*fichier*]
: Calcule et/ou vérifie la somme “MD5” d'un fichier ou flux. Voir RFC 1321.

man [n] *commande*

Affiche la page de manuel (aide en ligne) pour la commande. L'argument n permet de spécifier le numéro de la section de manuel (utile lorsque la commande existe dans plusieurs sections). Les numéros des sections sont : 1 (commandes utilisateur), 2 (appels systèmes), 3 (fonctions librairies C), 4 (devices), 5 (formats de fichiers), 6 (jeux), 7 (divers), 8 (administration) et n (new, programmes locaux).

Voir aussi le lexique page 72.

more [*fichier*]
: Affiche un fichier page par page (aide en ligne, taper 'h'). Si aucun fichier n'est spécifié, lit sur l'entrée standard.

less [*fichier*]
: *less is more*. Comme more, avec plus de fonctionnalités (taper 'h' pour en savoir plus).

tail [+n | -n] [*fichier*]
: La forme `tail +n` permet d'afficher un fichier à partir de la ligne n. La forme `tail -n` affiche les n dernières lignes du fichier. Si aucun fichier n'est spécifié, lit sur l'entrée standard.

tar options [*fichier ou répertoire*]
: La commande `tar` permet d'archiver des fichiers ou une arborescence de fichiers, c'est à dire de les regrouper dans un seul fichier, ce qui est très pratique pour faire des copies de sauvegardes d'un disque, envoyer plusieurs fichiers en une seul fois par courrier électronique, etc.

Pour créer une nouvelle archive, utiliser la forme

```
$ tar cvf nom-archive repertoire
```

qui place dans le nouveau fichier "nom-archive" tous les fichiers situés sous le répertoire indiqué. On donne généralement l'extension `.tar` aux fichiers d'archives.

Pour afficher le contenu d'une archive, utiliser

```
$ tar tvf nom-archive
```

Pour extraire les fichiers archivés, utiliser

```
$ tar xvf nom-archive
```

les fichiers sont crées à partir du répertoires courant.

Nombreuses autres options. Notons que les noms de fichiers peuvent être remplacés par `-` pour utiliser l'entrée ou la sortie standard (filtres), comme dans les exemples ci-dessous :

– Archivage d'un répertoire et de ses sous-répertoires dans un fichier `archive.tar` :

```
$ tar cvf archive.tar repertoire
```

– Archivage et compression au vol :

```
$ tar cvf - repertoire | gzip > archive.tar.gz
```

– Pour afficher l'index de l'archive ci-dessus :

```
$ zcat archive.tar.gz | tar tvf -
```

– l'option `z` permet de (dé)comprimer directement. On utilse dans ce cas l'extension `.tgz` :

```
$ tar cvfz  archive.tgz repertoire
```

– Copie complète récursive d'un répertoire `repert` dans le répertoire `destination` :

```
$ tar cvf - repert | (cd destination; tar xvfp -)
```

Les parenthèses sont importantes : la deuxième commande `tar` s'exécute ainsi dans le répertoire de destination.
Cette façon de procéder est supérieure à `cp -r` car on préserve les propriétaires, droits, et dates de modifications des fichiers (très utile pour effectuer des sauvegardes).

uncompress [*fichier*]
: Décompresse un fichier (dont le nom doit terminer par .Z) compressé par `compress`.

wc [-cwl] [*fichier* ...]
: Affiche le nombre de caractères, mots et lignes dans le(s) fichier(s). Avec l'option -c, on obtient le nombre de caractères, avec -w, le nombre de mots (*words*) et avec -l le nombre de lignes.

which *commande*
: Indique le chemin d'accès du fichier lancé par "commande".

who [am i]
: Liste les utilisateurs connectés au système. La forme `who am i` donne l'identité de l'utilisateur.

zcat [fichiers]
: Similaire à `cat`, mais décompresse au passage les fichiers (ou l'entrée standard) compressés par `compress`.

uuencode [fichier] nom
: Utilisé pour coder un fichier en n'utilisant que des caractères ascii 7 bits (codes entre 0 et 127), par exemple pour le transmettre sur un réseau ne transmettant que les 7 bits de poids faible.

`uudencode` lit le fichier (ou l'entrée standard si l'on ne précise pas de nom) et écrit la version codée sur la sortie standard. Le paramètre `nom`

précise le nom du fichier pour le décodage par `uudecode`.

uudecode [fichier]
Décode un fichier codé par `uuencode`. Si l'argument `fichier` n'est pas spécifié, lit l'entrée standard. Le nom du fichier résultat est précisé dans le codage (paramètre `nom` de `uuencode`).

5.3 Filtres de texte

Les *filtres* sont des utilitaires qui prennent leurs entrées sur l'entrée standard, effectuent un certain traitement, et fournissent le résultat sur leur sortie standard. Notons que plusieurs utilitaires déja présentés peuvent être vus comme des filtres (head, tail).

grep [option] *motif* $fichier_1$ ... $fichier_n$
Affiche chaque ligne des fichiers $fichier_i$ contenant le motif *motif*. Le motif est une *expression régulière*.

Options :	-v	affiche les lignes qui ne contiennent *pas* le motif.
	-c	seulement le nombre de lignes.
	-n	indique les numéros des lignes trouvées.
	-i	ne distingue pas majuscules et minuscules.

sort [-rn] [*fichier*]
Trie les lignes du fichier, ou l'entrée standard, et écrit le résultat sur la sortie. L'option -r renverse l'ordre du tri, l'option -n fait un tri numérique.

tr [options] *chaine1 chaine2*
Recopie l'entrée standard sur la sortie standard en remplaçant tout caractère de *chaine1* par le caractère de position correspondante dans *chaine2*.

uniq [-cud] *fichier*
Examine les données d'entrée ligne par ligne et détermine les lignes dupliquées qui sont consécutives : -d permet de retenir que les lignes dupliquées. -u permet de retenir que les lignes non dupliquées. -c permet de compter l'indice de répétition des lignes.

5.4 Manipulation des processus

kill -*sig PID*

Envoie le signal *sig* au processus de numéro *PID*. Voir la table 8.1 (page 66) pour une liste des signaux. *sig* peut être soit le numéro du signal, soit son nom (par exemple `kill -STOP 1023` est l'équivalent de `kill -19 1023`).

ps [-e][-l]

Affiche la liste des processus.
L'option -l permet d'obtenir plus d'informations. L'option -e permet d'afficher les processus de tous les utilisateurs.
ps affiche beaucoup d'informations. Les plus importante au début sont :
- `UID` : identité du propriétaire du processus ;
- `PID` : numéro du processus ;
- `PPID` : PID du père du processus ;
- `NI` : priorité (nice) ;
- `S` : état du processus (R si actif, S si bloqué, Z si terminé).

nice [-*priorite*] *commande*

Lance l'exécution de la commande en modifiant sa . Permet par exemple de lancer un processus de calcul en tâche de fond sans perturber les processus interactifs (éditeurs, shells etc.), en lui affectant une priorité plus basse.
Le noyau accorde moins souvent le processeur aux processus de basse priorité.

renice *priorite* -p pid

Modifie la priorité d'un processus.

5.5 Gestion des systèmes de fichiers

5.5.1 Principes

Tous les disques, disquettes et CD-ROMs connectés à un ordinateur sous UNIX sont accédés via une arborescence unique, partant du répertoire racine `/`. La complexité du système est ainsi masquée à l'utilisateur, qui peut utiliser les mêmes commandes pour accéder à un fichier sur disque dur ou sur CD-ROM.

Remarquez que ce principe est différent de celui employé par les systèmes MS-DOS et Windows, pour lesquels chaque *volume* (disque) possède une racine spécifique repérée par une lettre (`A:\`, `C:\`, etc).

Bien entendu, les fichiers sont organisés de façon différentes sur les disques durs, les disquettes, ou les CD-ROMS : chaque périphérique possède son propre *système de fichiers*.

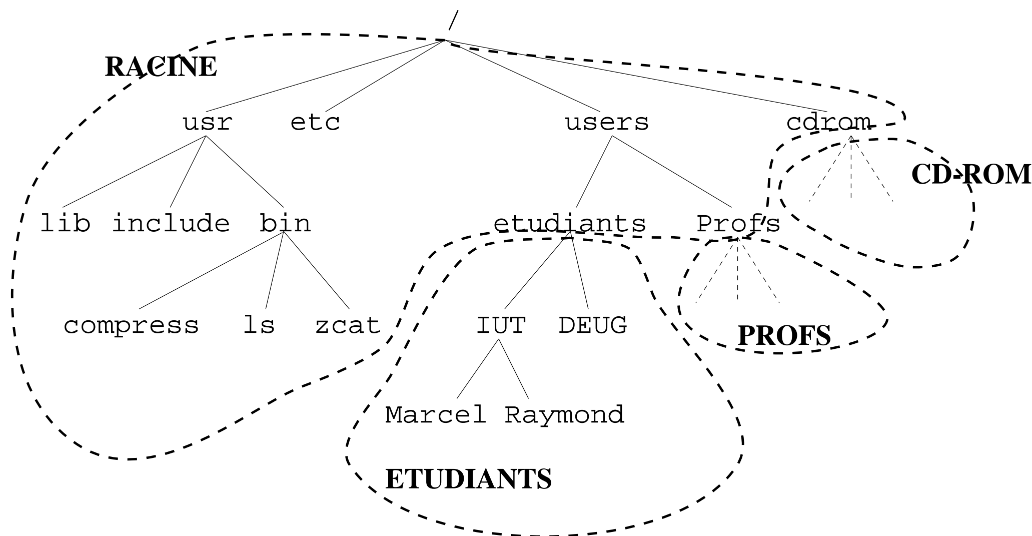


FIGURE 5.1 – Dans cet exemple, l'arborescence des fichiers est composée de quatre *systèmes de fichiers* : le système racine (point de montage /, deux systèmes pour les utilisateurs (/users/etudiants et /users/profs), et un système pour le CD-ROM (/cdrom).

Les différents systèmes de fichier, que l'on peut considérer comme des sous-arborescences associées à un périphérique spécial, sont *montées* sur l'arborescence racine. Chaque système de fichier doit posséder un point de montage, qui est au départ (avant montage) un simple répertoire vide (voir figure 5.1).

La commande `mount` permet d'associer au point de montage le système de fichier correspondant. Il faut lui préciser le pilote de périphérique (*device driver*) utilisé pour accéder à ce système. Nous n'étudions pas dans ce cours la gestion des pilotes de périphériques sous UNIX ; notons simplement que chaque pilote correspond à un pseudo-fichier dans le répertoire système `/dev`.

5.5.2 Commandes de base

Nous décrivons simplement trois commandes permettant d'observer l'état des systèmes de fichiers. Nous de décrivons pas leur utilisation pour administrer le système (ajouter un système de fichier, etc).

mount [*point d'entree*]

Permet d'afficher la liste des sytèmes de fichiers en service (montés). Le format d'affichage dépend légèrement de la version d'UNIX utilisée. Un exemple simple sous Linux :

```
$ mount
/dev/sda3 on / type ext2 (rw)
/dev/sda4 on /users type ext2 (rw)
/dev/sdb1 on /jaz type ext2 (rw)
```

On a ici trois systèmes de fichiers, gérés par les pilotes `/dev/sda3`, `/dev/sda4` et `/dev/sdb1` (trois disques durs SCSI), et montés respectivement sur la racine, `/users` et `/jaz`.

La commande `mount` est aussi utilisée pour monter un nouveau système de fichiers dans l'arborescence (lors du démarrage du système ou de l'ajout d'un disque).

df [*chemin*]

`df` (*describe filesystem*) affiche la capacité totale d'un système de fichiers (généralement en KiloOctets), le volume utilisé et le volume restant disponible.

Le format d'affichage dépend aussi de la version utilisée. Exemple sous Linux :

```
$ df /users
Filesystem  1024-blocks  Used Available Capacity Mounted on
/dev/sda4       417323  347789    47979     88%   /users
```

On a ici 417323 Ko au total, dont 347789 sont utilisés et 47979 libres.

du [-s] [*chemin*]

`du` (*disk usage*) affiche la taille occupée par le fichier ou répertoire spécifié par l'argument *chemin*. Avec l'option `-s`, n'affiche pas le détail des sous-répertoires.

Exemple :

```
$ du POLYUNIX
10      POLYUNIX/fig
19      POLYUNIX/ps
440     POLYUNIX
$ du -s POLYUNIX
440     POLYUNIX
```

Le répertoire `POLYUNIX` occupe ici 440 Ko.