Carnet de Travaux Pratiques
Système et programmation système
Licence 2 Informatique

Julien Bernard Eric Merlet

Introduction

Ces travaux pratiques vous permettent de vous entraîner à manipuler toutes les notions vues en cours. Si vous ne parvenez pas à finir le TP dans le temps encadré imparti, il sera obligatoire de le finir pour la semaine suivante.

Tout le travail demandé ne nécessite qu'un éditeur de texte (comme `Kate`) et un terminal pour le shell (comme `Konsole`). En particulier, il est expressément interdit d'utiliser un gestionnaire de fichier graphique (comme `Konqueror`).

La langue de la programmation est l'anglais. Vous devrez donc écrire tout votre code (commentaires inclus) en utilisant l'anglais. Cela concerne aussi bien les noms (variables, fonctions, etc) dans votre code que l'affichage.

Table des matières

Travaux Pratiques de Système n°1

Exercice 1 : Prise en main de l'environnement

Cet exercice doit vous permettre de prendre en main votre environnement de travail. Pour cela, connectez-vous d'abord sur votre compte avec votre nom d'utilisateur. Puis, ouvrez une console (`Konsole` par exemple) qui vous donne accès à un interpréteur de commande.

→ Connexion à votre compte

Avant de vous connecter sur votre compte, tapez CTRL+ALT+F1. Entrez votre nom d'utilisateur et votre mot de passe.

Question 1.1 Dans quel répertoire vous trouvez-vous ?

Question 1.2 À l'aide de la commande `ls(1)`, afficher le contenu de votre répertoire. Combien de fichiers et répertoires cachés avez-vous ?

Question 1.3 Effacer l'écran à l'aide de la commande `clear` ou avec la combinaison de touches CTRL+L.

Question 1.4 Fermer la session à l'aide de la commande `exit` ou avec la combinaison de touche CTRL+D.

Pour revenir à l'écran de login graphique, tapez CTRL+ALT+F7.

→ Commandes de base

Question 1.5 Tester les commandes vues en cours : `whoami(1)`, `uname(1)`, `uptime(1)`, `date(1)`, `cal(1)`, `echo(1)`, `man(1)`, `whatis(1)`, `apropos(1)`. En particulier, en cas d'options multiples, vous pouvez utiliser deux écritures : `-a -b -c` ou `-abc`. Le vérifier à l'aide de la commande `uname` par exemple.

Question 1.6 Comment obtenir une commande équivalente à `whoami` avec la commande `id` ?

Question 1.7 Dans quels sections pouvez-vous trouver une manpage appelée `time` ?

→ Système de fichier

Pour créer un fichier, il existe la commande `touch(1)`. En fait, cette commande met à jour le `atime` et le `mtime` d'un fichier (en le touchant), mais si le fichier n'existe pas, il est créé. Vous utiliserez cette commande pour créer des fichiers vide. Pour rappel, la commande pour créer un répertoire est `mkdir`.

En outre, pour les commandes `cp(1)`, `rm(1)`, `mv(1)`, vous testerez l'option `-i` qui permet de demander une confirmation.

Question 1.8 Créer un répertoire `SYS` dans votre répertoire utilisateur. Entrer dans ce répertoire puis créer un répertoire `exemple`. Entrer dans le répertoire `exemple`.

Question 1.9 Dans le répertoire `exemple` que vous venez de créer, créer les répertoires `skywalker/luke` en une seule commande. Puis créer un répertoire `skywalker/anakin` et un fichier `skywalker/anakin/README`.

Question 1.10 Supprimer le répertoire `skywalker/luke`. Que se passe-t-il si vous essayez de supprimer `skywalker/anakin` ?

Question 1.11 Renommer (en fait, déplacer) le répertoire `skywalker/anakin` en `darth_vader`.

Question 1.12 Créer un répertoire `yoda` et un fichier `yoda/README`. Copier le fichier `yoda/README` dans le fichier `yoda/README.old`. Créer un lien physique appelé `yoda/README2` sur l'inode du fichier `yoda/README`. Ecrire la chaîne de caractères "bonjour" dans `yoda/README`. Afficher le contenu de `yoda/README2`. Visualiser les numéros d'inode et le nombre de liens physiques (pointant sur les inodes) des fichiers `yoda/README` et `yoda/README2`.

Question 1.13 Créer un répertoire `.private` dans `yoda`. Créer un lien symbolique nommé `LREADME` dans `.private` sur le fichier `yoda/README`. Afficher le contenu de `yoda/.private/LREADME`.

Question 1.14 Supprimer le lien physique `yoda/README2`. Un fichier sur disque a-il été supprimé ? Supprimer `yoda/.private/LREADME`. Un fichier sur disque a-il été supprimé ? Supprimer le lien physique `yoda/README`. Un fichier sur disque a-il été supprimé ?

Question 1.15 Supprimer récursivement le répertoire `yoda` (c'est-à-dire le répertoire et tout ce qu'il contient) à l'aide de l'option `-r` de `rm`.

Continuez à créer des fichiers et des répertoires et à les déplacer, copier, supprimer.

Exercice 2 : Archives

Cet exercice consiste à manipuler des archives (compressées ou non) à l'aide de la commande `tar(1)` (*Tape ARchive*). À l'origine, la commande `tar(1)` servait à créer des sauvegardes sur des bandes magnétiques (*tape*). Les bandes magnétiques avaient une plus grosse capacité de stockage que les disques durs mais avait un accès linéaire (c'est-à-dire que le temps pour accéder à une donnée était fonction de sa position sur la bande). La commande `tar(1)` a évolué pour créer des archives dans des fichiers.

La commande `tar(1)` prend en option :

- Une option de compression parmi :
  - aucune option si on ne veut pas de compression
  - `z` pour compresser avec `gzip`

  - `j` pour compresser avec `bzip2`
- Une action parmi :
  - `c` pour créer une archive avec des fichiers
  - `x` pour extraire les fichiers d'une archive
  - `t` pour lister le contenu d'une archive
  - `r` pour ajouter des fichiers dans une archive
  - `u` pour mettre à jour des fichiers dans une archive
- L'option `f` qui indique qu'on utilise un fichier dont le nom est indiqué
- L'option `v` (non-obligatoire) si vous voulez afficher le déroulement des opérations

Par exemple :

```
tar cf archive.tar repertoire
tar cf archive.tar fichier1 fichier2
```

Question 2.1 Créez une archive `exemple.tar` avec le répertoire `exemple` de l'exercice de prise en main.

Question 2.2 Allez chercher le fichier des figures sur MOODLE. Quels sont les fichiers contenus dans cette archive ?

Question 2.3 Extrayez l'archive puis créez une archive `inodes.tar.bz2` avec les trois figures concernant les inodes.

Question 2.4 Les options `z` et `j` sont en fait équivalentes à appeler directement `gzip(1)` et `bzip2(1)` (pour la compression) ou `gunzip(1)` et `bunzip2(1)` (pour la décompression). Décompressez l'archive `inodes.tar.bz2` sans en extraite les fichiers. Vous obtenez l'archive `inodes.tar`.

Question 2.5 Ajoutez les figures concernant la compilation à l'archive `inodes.tar` obtenue à la question précédente.

Exercice 3 : Somme des longueurs des arguments d'un script

Le but est de faire un script `arguments.sh` qui détermine et affiche :
- le nombre d'arguments d'un script
- la somme des longueurs des arguments d'un script

Indices :
→ `$#`
→ `${#VAR}` donne la longueur en nombre de caractères de la valeur de la variable `VAR`

Voici un exemple d'exécution :

```
$ ./arguments.sh aa bbb cc
Nombre d'arguments = 3
Somme des longueurs des arguments = 7
$
```

Question 3.1 Ecrire un script qui détermine et affiche le nombre d'arguments transmis.

Question 3.2 Compléter le script pour calculer et afficher la somme des longueurs des arguments.

Exercice 4 : Nombre de fichiers du répertoire courant

Le but est de faire un script `fichiers.sh` qui détermine et affiche :
— le nombre de fichiers non cachés du répertoire courant
— le nombre de fichiers cachés du répertoire courant
Voici un exemple d'exécution :

```
$ ls -a
.  ..  arguments.sh  .bashrc  fic1  fichiers2.sh  fichiers.sh  toto
$ ./fichiers.sh
Nombre de fichiers non cachés du répertoire courant = 5
Nombre de fichiers cachés du répertoire courant = 3
$
```

Question 4.1 Ecrire un script qui détermine et affiche le nombre de fichiers non cachés du répertoire courant.

Question 4.2 Compléter le script pour déterminer et afficher le nombre de fichiers cachés du répertoire courant.

Travaux Pratiques de Système n°2

Exercice 5 : Jeu du «Plus petit / Plus grand»

Le but est de faire un script `plus_moins.sh` pour jouer au jeu «Plus petit / Plus grand». Ce jeu consiste à deviner un nombre entre 1 et 100 choisi au hasard par l'ordinateur. Pour cela, le joueur peut faire plusieurs propositions et l'ordinateur dit si la valeur qu'il a choisie est plus petite ou plus grande que la proposition du joueur.

Voici un exemple d'utilisation du jeu :

```
L'ordinateur a choisi une valeur entre 1 et 100.
Quelle est votre proposition ? 50
La valeur est plus petite
Quelle est votre proposition ? 20
La valeur est plus grande
Quelle est votre proposition ? 35
Gagné ! Vous avez trouvé en 3 essais
```

Question 5.1 Définir une variable appelée `MAX` avec la valeur 100. Faire tirer à l'ordinateur une valeur au hasard entre 1 et `MAX` en utilisant la commande :
`$ hexdump -n 2 -e '/2 "%u"' /dev/urandom`
Cette commande lit 2 octets (`-n 2`) dans le fichier virtuel `/dev/urandom` et affiche sur la sortie standard le résultat de la conversion binaire vers décimale des 2 octets lus (`-e '/2 "%u"'`).

Question 5.2 Demander à l'utilisateur sa proposition tant qu'il n'a pas trouvé la bonne réponse et afficher le résultat par rapport à la valeur de l'ordinateur. Indice : `test(1)`.

Question 5.3 Afficher le nombre d'essais nécessaires pour trouver la réponse

On veut limiter le nombre d'essais possibles pour le joueur à 6. Si le joueur échoue 6 fois de suite, l'ordinateur affiche un message au joueur. Par exemple :

```
Perdu ! La bonne réponse était : 32
```

Question 5.4 Modifier le programme pour s'arrêter si le joueur fait 6 propositions perdantes et afficher un message adéquat

Exercice 6 : Lister les fichiers d'un répertoire en shell

Le but est de faire un script `ls.sh` qui est une version très simple de `ls(1)` en shell. Le script prendra éventuellement un seul argument, le nom du répertoire à lister (le script liste le répertoire courant par défaut).
`Usage : ./ls.sh [dir]`

Question 6.1 Vérifier la validité de l'éventuel argument transmis et déterminer le répertoire à lister.

Question 6.2 Parcourir tous les fichiers du répertoire courant et afficher leur nom. Indice : `for`

Question 6.3 Déterminer le type de chaque fichier et afficher cette information dans le cas où c'est un fichier régulier ou un répertoire. Indice : `test(1)`

Exercice 7 : Recherche des IP attribuées sur un réseau local

Le but est de faire un script `ping.sh` qui détermine les adresses IPv4 attribuées dans une plage d'adresses.
Les adresses IPv4 sont codées sur 32 bits. Elles sont usuellement représentées en utilisant une notation décimale à points où chaque nombre décimal correspond à un octet.
Exemple : `172.16.128.15`
Les adresses IP sont hierarchiques :

- une partie de l'adresse située à gauche désigne le réseau auquel elle appartient
- la partie située à droite désigne l'hôte sur le réseau

Dans nos salles de TP, la partie "réseau" de l'adresse est codée sur les 3 premiers octets (octets de poids fort) et la partie "hôte" sur un seul octet. Ainsi, dans l'exemple précédent, la partie "réseau" de l'adresse est égale à `172.16.128`, et la partie "hôte" à `15`.

Question 7.1 Déterminer l'IP de votre machine en utilisant la commande `ifconfig(1)`. En déduire sa partie "réseau".

La commande `ping(1)` permet, grâce à l'envoi de paquets, de vérifier si une machine distante répond et, par extension, si elle est accessible via le réseau.

Question 7.2 Essayer de "pinguer" votre propre adresse IP, ainsi que celle de votre voisin. (utiliser `CTRL^C` pour tuer le processus qui exécute la commande `ping(1)`).

Question 7.3 Déterminer les options de la commande `ping(1)` qui permettent d'envoyer un seul paquet avec une deadline de une seconde (si l'hôte distant ne répond pas).

Le synopsis de lancement du script `ping.sh` est le suivant :
`./ping.sh partieRéseau partieHoteBasse partieHoteHaute`
Exemple : `./ping.sh 172.16.128 10 50`
Dans le cas précédent, le script doit tester si les IP comprises entre 172.16.128.10 et 172.16.128.50 (bornes incluses) répondent.
Les arguments `partieHoteBasse` et `partieHoteBasse` doivent être des entiers compris dans l'intervalle $[1, 254]$ et avec `partieHoteBasse` $\geq$ `partieHoteBasse`

Quelques exemples d'exécution :

```
$ ./ping.sh 194.57.88 170 168
usage : ./ping.sh partieRéseau partieHoteBasse partieHoteHaute
Erreur : partieHoteBasse et partieHoteHaute doivent être des entiers dans [1, 254]
         avec partieHoteBasse <= partieHoteHaute

$ ./ping.sh 194.57.88 170 180
Adresses IP attribuées sur le réseau 194.57.88.0 pour dans la plage [.170, .180]:
194.57.88.170
194.57.88.178
```

```
--> Nombre d'IP attribuées : 2
$
```

Question 7.4 Vérifier la présence et la validité des arguments. En cas d'erreur, sortir et afficher un message indiquant l'usage du script (synopsis) et l'origine de l'erreur.

Question 7.5 Compléter le script pour déterminer les IPs attribuées, ainsi que leur nombre.

Exercice 8 : Destruction récursive des répertoires vides

Le but est de faire un script `rec_rmdir.sh` qui efface récursivement les répertoires vides du répertoire courant. Pour cela, on peut définir une fonction nommée `rec_rmdir` qui fera tout le travail et uniquement appeler cette fonction avec le répertoire courant :

```
rec_rmdir .
```

Question 8.1 Dans la fonction `rec_rmdir`, vérifier qu'il y a un seul paramètre et l'afficher, sinon sortir de la fonction (avec la commande `return`). Indice : `$#`

Question 8.2 Vérifier que le paramètre est bien un répertoire, puis, si c'est le cas, entrer dans ce répertoire. Indice : `test(1)`.

Question 8.3 Parcourir tous les fichiers de ce répertoire et pour chaque fichier qui est lui-même un répertoire, afficher son nom. Indice : `for`

Arrivé à ce stade, il ne reste plus qu'à appeler récursivement la fonction `rec_rmdir` sur le répertoire trouvé puis de le supprimer (avec `rmdir(1)`).
Seulement, si la fonction s'appelle récursivement sans précaution, la variable utilisée pour itérer sur les fichiers sera écrasée : en effet, par défaut, toutes les variables définies dans une fonction sont visibles globalement (pour le processus en cours) et donc ne sont pas uniquement locales à la fonction. Pour résoudre ce problème, il faut soit créer un sous shell (nouveau processus) à chaque appel de la fonction, soit déclarer que la variable utilisée pour itérer est locale à la fonction (cf. scripts exemples sur Moodle).
Pour créer un sous shell, on peut entourer le groupe de commandes concerné par des parenthèses (semblables aux accolades). Deux manières de faire sont possibles :

- soit placer les parenthèses autour de l'appel récursif uniquement
- soit placer les parenthèses à la place des accolades qui délimitent le corps de la fonction

Question 8.4 Terminer le script avec les indications données. On veillera notamment à supprimer le message d'erreur de `rmdir(1)` quand le répertoire n'est pas vide.

Question 8.5 Bonus : supprimer les fichiers vides