
TP d'initiation au *shell* Unix¹ – Correction

Le thème de ce TP est l'utilisation du *shell* en mode interactif avec l'interpréteur `bash`. L'invite est représentée par le symbole `$`.

1 Instructions simples

1.1 Comment accéder à l'aide du *shell* ?

1.1.1 `less`

Le filtre `less` est un filtre de visualisation qui retourne dans le terminal — c'est-à-dire sur sa sortie standard — les octets constituant le fichier qui lui est passé en argument sur la ligne de commande. Ainsi, on peut examiner le contenu du fichier `/etc/bash.bashrc` par l'instruction suivante :

```
$ less /etc/bash.bashrc
```

Ce filtre est interactif. Pour sortir, il suffit de presser la touche `Q`. De plus, en pressant la touche `H`, on affiche l'ensemble des raccourcis claviers disponibles.

Téléchargez le fichier `romeoandjuliet.txt` :

```
$ wget http://www.math.jhu.edu/~jkramer/shakespeare/romeoandjuliet.txt
```

Question. Utilisez les fonctionnalités de recherche du filtre `less` pour trouver toutes les occurrences du mot “Juliet” dans ce texte.

Commandes `/` (pour chercher un motif) et `n` (pour chercher l'occurrence suivante du motif).

1. Merci à Alexandre Sedoglavic !

1.1.2 man

En théorie, chaque filtre devrait posséder une page d'aide accessible par le biais du filtre `man`. L'ensemble des filtres disponibles sur une machine sont également documentés en ligne².

Par défaut, la sortie standard de la commande `man` est redirigée dans l'entrée standard du visualisateur (*pager* en anglais) `less`. Les raccourcis claviers de `less` permettent d'exploiter facilement les informations incluses dans le manuel.

Question. En utilisant `man` et les raccourcis clavier de `less`, répondez aux questions suivantes.

1. Quelle option permet de spécifier à `man` quel *pager* utiliser ?

L'option `-P`.

2. Que permet l'option `-k` du `man` ?

Cette option permet de visualiser l'ensemble des pages contenant un mot-clef. Équivalent à la commande `apropos`.

1.2 Manipulations du système de fichiers

1.2.1 Création d'arborescence

Récupérez les PDF du cours et du TP un utilisant les commandes (sans saut de ligne) :

```
$ wget  
www.grappa.univ-lille3.fr/~champavere/Enseignement/0910/ls4info/shell-cm.pdf
```

```
$ wget  
www.grappa.univ-lille3.fr/~champavere/Enseignement/0910/ls4info/shell-tp.pdf
```

1. Reproduisez l'arborescence ci-dessous :
Shell -> Cours -> shell-cm.pdf
 -> TP -> shell-tp.pdf
 -> Personnel
en utilisant les commandes `cd`, `mkdir`, `mv`.

2. En anglais ou en français.

```
$ mkdir Shell
$ cd Shell
$ mkdir Cours
$ mkdir TP
$ mkdir Personnel
$ mv ../shell-cm.pdf Cours
$ mv ../shell-tp.pdf TP
```

2. Modifiez les droits du répertoire `Personnel` afin d'être le seul à pouvoir y accéder en utilisant la commande `chmod`.

```
$ chmod 700 Personnel
```

1.2.2 Création et manipulation d'archives

Nous allons archiver l'arborescence que vous venez de créer. Par convention, une archive se termine par le suffixe `tar` (`tgz` si elle est compressée).

1. Utilisez la commande `tar cvf <nom_archive> <nom_repertoire>` pour faire une archive de votre répertoire.
2. Utilisez la commande `tar cvfz <nom_archive> <nom_repertoire>` pour faire une archive compressée de votre répertoire.
3. Listez le contenu de l'archive non compressée (commande `tar tvf <nom_archive>`) (idem pour l'archive compressée).
4. Extrayez de l'archive non compressée le fichier `shell-cm.pdf` (commande `tar xvf <nom_archive> <fichier_a_extraire>`). Attention, le chemin d'accès à ce fichier doit être indiqué.
5. Ajoutez un fichier à l'archive en utilisant l'option `r`.

```
$ tar -r -file=<nom_archive> <nom_fichier>
```

6. Décompressez et ouvrez l'archive compressée afin d'obtenir son contenu (commande `tar xvfz <nom_archive> <nom_repertoire>`).
7. Pour les utilisateurs de Windows qui veulent utiliser `zip`, recommencez les exercices ci-dessus avec ce filtre comme outils de manipulation d'archive (voir `man zip`).

1.2.3 Liens symboliques

Nous allons lier symboliquement un fichier `foo` sur votre compte avec l'exécutable `ls` se trouvant dans le répertoire `/bin/`. Pour ce faire, utilisez la commande :

```
$ ln -s /bin/ls foo
```

Question. Exécutez le fichier que vous venez de créer. Quelle est sa taille en octets? Déduisez s'il s'agit d'une copie ou d'un lien symbolique.

1.2.4 Liens physiques (optionnel)

Vous avez peut-être déjà été surpris par le fait que nos voisins flamands s'obstinent — outre à parler une langue incompréhensible — à affubler certaines villes de noms impossibles. Par exemple :

Nom français	Nom flamand
Anvers	Antwerpen
Bruxelles	Brussel
Gand	Gent
La Haye	Den Haag
Lille	Rijsel
Paris	Paris
Tournai	Doornik

Nous allons profiter de cet état de fait pour nous exercer à faire des liens physiques (*hard link*). Ces liens permettent le partage de fichiers sans duplication de ces derniers (`man ln`).

1. Construisez un répertoire `villes` contenant un fichier pour chaque nom de ville en français (`man touch`).

```
$ mkdir villes
$ for v in Anvers Bruxelles Gand La_Haye Lille Paris Tournai ; do
\
> touch villes/$v ; \
> done
```

2. Construisez un répertoire `steden` contenant un lien physique pour chaque nom de ville en flamand et qui pointe vers le fichier correspondant dans le répertoire `villes`.

```
$ mkdir steden
$ ln villes/Anvers steden/Antwerpen
$ ln villes/Bruxelles steden/Brussel
...
```

1.3 Manipulation de processus depuis le *shell*

Considérons les commandes suivantes :

```
$ xterm; emacs
```

Si vous sortez du *shell* lancé dans le terminal X (**xterm**), l'éditeur **emacs** est lancé. Les processus sont exécutés séquentiellement : dès que l'un a terminé, le suivant commence. On peut lancer plusieurs processus de manière concurrente (ils sont en concurrence pour l'utilisation du processeur). Pour ce faire, on utilise l'opérateur **&** :

```
$ xterm & emacs &
```

Après avoir lancé la commande ci-dessus :

1. en utilisant la commande **jobs**, déterminez le numéro du *job* correspondant à **emacs** ;

```
$ jobs -l
[2]+ 19132 Running          emacs &
```

2. en utilisant la commande **fg**, passez **emacs** en mode séquentiel ;
3. en utilisant le raccourci clavier **Ctrl** + **Z** et la commande **bg**, passez **emacs** en mode concurrent ;
4. supprimez ce processus en utilisant la commande **kill**.

```
$ kill 19132

Alternative :
$ kill %2
```

Question. De quel type sont les commandes **jobs**, **fg**, **bg** et **kill** ?

Ce sont des commandes internes du *shell* :

```
$ type jobs
jobs est une primitive du shell
```

1.4 Redirection

1.4.1 Éditer avec **cat** (optionnel)

Sans arguments, la commande **cat** retourne son entrée standard vers sa sortie standard jusqu'à réception du caractère EOF (fin de fichier, équivaut à **Ctrl** + **D**).

1. Utilisez une redirection pour mettre un vers dans le fichier **/tmp/foo**.

```
$ cat > /tmp/foo << FIN
> Un vers, ça va
> FIN
```

2. Toujours avec la commande `cat`, affichez votre vers.

```
$ cat /tmp/foo
Un vers, ça va
```

3. Réutilisez la même méthode pour ajouter un second vers au fichier `/tmp/foo` sans détruire le premier.

```
$ cat >> /tmp/foo << FIN
> Deux vers, bonjour les dégâts
> FIN
```

Alternative :

```
$ echo "Deux vers, bonjour les dégâts" >> /tmp/foo
```

4. On cherche à faire une copie `/tmp/bar` du fichier `/tmp/foo` sans utiliser la commande `cp`. Comment faire ?

```
$ cat < /tmp/foo > /tmp/bar
```

5. Que fait la commande suivante ?

```
$ cat < /tmp/foo > /tmp/foo
```

Elle vide le fichier `/tmp/foo`.

1.4.2 Quizz

1. Que contient le fichier `/tmp/foo` après l'exécution de la commande suivante (sans saut de ligne) ?

```
$ (echo "Début" > /tmp/foo; ls -al > /tmp/foo) || echo "Fin" >>
/tmp/foo;
```

Le résultat de la commande `ls -al`.

Comment modifier cette commande afin que le fichier `/tmp/foo` commence par `Début`, contienne la liste des fichiers du répertoire courant et se termine par `Fin` (en rajoutant 1 caractère et en modifiant exactement 2 caractères) ?

Il y a 2 modifications à apporter, changer le OU en ET (ou en `;`) et ne pas écraser le fichier `/tmp/foo` après sa création. Ceci donne :

```
$ (echo "Début" > /tmp/foo; ls -al >> /tmp/foo) && echo "Fin" >>
/tmp/foo;
```

2. Que font les commandes ci-dessous ?

```
$ ls /bin/fichierquinexistepas /bin/ 1> /tmp/foo1 2>/tmp/foo2
$(ls /ntn /bin 1>&2) 2> /tmp/foo
$ find / -perm +a+x -exec grep -IH "^#\!" {} \; 2> /dev/null >
/tmp/foo
```

- 1> redirige la sortie standard ;
- 2> redirige la sortie d'erreur ;
- 1>&2 duplique la sortie standard vers la sortie d'erreur ;

La redirection vers `/dev/null` envoie la sortie vers un périphérique particulier qui "oublie" toute donnée qui y est écrite.

1.4.3 Tube de communication (optionnel)

Question. Écrire une commande qui donne les noms de tous les utilisateurs ayant au moins un processus en cours d'exécution.

Indications :

- Avec l'option `aux`, la commande `ps` affiche les mêmes informations pour tous les utilisateurs connectés à la machine.
- La commande `tail --lines=+2` ne garde que les dernières lignes d'un fichier à l'exception des 2 premières (si la version de `tail` dont vous disposez ne suit pas cette syntaxe, consultez la page correspondante du manuel).
- La commande `cut` permet de supprimer des colonnes d'un fichier ; ces colonnes sont définies par le séparateur `s` fourni par l'option `-d's'` et on utilise l'option `-fm-n` pour ne garder que les colonnes allant de `m` à `n` incluses.
- La commande `sort` trie le contenu d'un fichier et la commande `uniq` élimine les lignes dupliquées dans un fichier trié.

```
$ ps aux | tail -lines=+2 | cut -f1 -d' ' | sort | uniq
```

1.5 Quelques commandes et précisions utiles

1.5.1 Historique des commandes (optionnel)

Le *shell* gère une liste des dernières commandes exécutées ; la taille de cette liste est donnée par la variable `shell HISTSIZE`.

Si cette variable est initialisée à 40, ce mécanisme permet d'obtenir un historique des 40 dernières commandes exécutées et de redemander l'exécution de ces commandes. La commande `history` permet d'obtenir l'historique des dernières commandes avec un numéro d'ordre associé à chaque commande. Les commandes ainsi numérotées peuvent être réexécutées simplement :

`!!` permet de réexécuter la dernière commande ;

`!n` permet de réexécuter la commande numéro `n` ;

`!ba` permet de réexécuter la commande la plus récente commençant par `ba`.

Testez la commande `history` et la réexécution de commandes selon les trois méthodes précisées ci-dessus.

Combien de commandes votre historique peut-il mémoriser ? Cette commande est-elle interne ou externe ? Pourquoi `!ls` est-il différent de `! ls`.

```
$ echo $HISTSIZE
500
$ type history
history est une primitive du shell
```

`!<motif>` exécute la commande la plus récente qui commence par `<motif>` ;

`! <commande>` inverse la valeur du code de sortie de `<commande>`.

1.5.2 Expressions régulières et métacaractères

1. Placez-vous dans le répertoire `/usr/bin` et, à l'aide des méta-caractères, listez par la commande `ls` (l'option `-d` de `ls` permet de lister les noms des répertoires plutôt que leur contenu) :
 - toutes les entrées dont le nom commence par `t` ;
 - toutes les entrées dont le nom comporte exactement trois caractères ;
 - toutes les entrées dont le nom comporte au moins trois caractères ;
 - toutes les entrées dont le nom comporte au plus trois caractères ;
 - toutes les entrées dont le nom comporte au moins un `d` ;
 - toutes les entrées dont le nom commence par `a` ou `t` ;
 - toutes les entrées dont le nom commence par `r`, `s`, `t` ou `u`.

Il y a beaucoup de réponses possibles à ces questions. Nous donnons des solutions illustrant certaines parties du cours :

```
$ ls t*
$ ls ???
$ ls ?*
$ ls ? ; ls ?? ; ls ???
$ ls *d*
$ ls [at]*
$ ls [r-u]*
```

2. En utilisant la commande `touch`, créer les fichiers `titi`, `toto` et `tutu` dans le répertoire `/tmp`. En utilisant la commande `rm` et des métacaractères, détruisez ces trois fichiers d'une seule commande.

```
$ rm /tmp/t[oui]t[oui]
```

1.5.3 La commande `find`

La commande `find` permet de retrouver des fichiers. En particulier, il est possible de rechercher des fichiers dont le nom correspond à un motif défini par une expression régulière.

Question. En utilisant la commande `find`, donnez la liste des fichiers dont le nom se termine par un tilde (`~`) (il s'agit de copies de sauvegarde engendrées par `emacs`).

```
$ find . -name "*~"
```

De plus, la commande `find` permet d'exécuter une commande. Donnez la commande permettant de trouver et d'effacer les fichiers inutiles dans votre arborescence.

```
Pour effacer les copies faites par les éditeurs, on peut utiliser  
find *~ -exec rm {} \;
```

2 Variables du *shell*

2.1 Code de retour (optionnel)

Expliquez les expressions suivantes :

```
$! ls; echo $?  
$ ls; echo $?  
$ (exit 3); echo $?
```

```
Cf. section 1.5.1.
```

2.2 La variable PATH

La variable PATH permet d'indiquer une liste de répertoires — séparés par le caractère : (code Ascii 58) — qui sont explorés lorsque l'interpréteur cherche un filtre externe (une variable CLASSPATH est utilisée dans le même esprit par le compilateur Java).

1. Affichez la valeur de votre variable PATH (c'est-à-dire évaluez cette variable).
2. Construisez un répertoire `mes_bin` dans votre répertoire de travail. Placez-y le lien symbolique `mon_ls` vers le filtre externe `/bin/ls`. Modifiez votre variable PATH de manière à ce que l'exécutable `mon_ls` soit utilisable depuis n'importe quel répertoire courant.

```
$ cd; mkdir mes_bin; ln -s /bin/ls mon_ls
$ PATH=$PATH:~/mes_bin
$ cd /tmp/; mon_ls
```

2.3 La variable PWD

Le *shell* dispose d'une variable PWD.

1. En vous inspirant des commandes suivantes :
\$ echo \$PWD; cd /bin; echo \$PWD
\$ cd /etc/init.d; echo \$PWD
indiquez ce que contient la variable PWD
2. Sans utiliser la commande `cd`, placez-vous dans le répertoire `/tmp`.
3. Sur le même principe, modifiez la variable HOME et faites interpréter la commande suivante :
\$ cd /bin; cd
À quoi correspond cette variable ?

1. La variable PWD contient le répertoire courant.
2. Il suffit d'affecter la valeur `/tmp` à la variable PWD.
\$ PWD=/tmp
3. Cette variable correspond à votre espace personnel et est utilisée par l'interprétation de la commande `cd`.

2.4 Exportation de variable

Cet exercice illustre l'export de variable.

1. Définissez une variable `F00` en lui affectant la chaîne de caractères `bar`.

```
$ F00=bar
```

2. Ouvrez un processus fils (`bash` ou `xterm`) et affichez le contenu de la variable `F00`. Que constatez-vous ?
3. Supprimez le processus fils. Exportez la variable `F00` par le biais de la commande interne `export` et recommencez l'opération précédente.

```
$ export F00=bar
```