Coquillages et crustacés : quelques commandes Shell

Objectifs

- Interpréteur de commandes : shell,
- Archivage et compression de données,
- Les commandes de gestion des répertoires et des fichiers,
- Redirections et tubes,
- La commande alias,
- Les variables d'environnement,
- Les fichiers de configuration du shell.

Attention : Le sujet étant assez dense et certaines notions plus avancées, nous avons distingué 2 niveaux de difficultés et/ou d'utilité ; nous souhaitons que vous abordiez toutes les notions de niveaux ∗.

1 Interpréteur de commandes : shell (niveau ∗)

Lorsqu'un utilisateur se connecte, un interpréteur de commandes (ou shell dans la terminologie unix/linux) est lancé [1]. Cet interpréteur a pour charge de recueillir les commandes de l'utilisateur et lancer leur exécution. Le shell exécute donc une boucle sans fin dont chaque pas consiste à :
- lire une ligne de commande (terminée par une frappe sur la touche Entrée),
- interpréter la ligne saisie : le shell propose en effet un certain nombre de raccourcis (jokers, alias...(voir plus loin)), destinés à faciliter la saisie des commandes. L'interprétation de la ligne va donc consister à remplacer les abréviations par la forme complète correspondante,
- lancer l'exécution de la commande.

Une ligne de commande se présente comme une suite de mots, séparés par des espaces. La forme générale d'une ligne de commande est :

```
chemin -options paramètres
```

où *chemin*, qui est le premier mot, est le chemin d'accès au fichier exécutable correspondant à la commande. Les options se distinguent (généralement) des paramètres en ce qu'elles sont précédées d'un tiret. Toutes les options et tous les paramètres sont séparés par des espaces.

Il existe deux grandes familles de shell : Bourne-Shell `sh` et C-Shell `csh` et leurs dérivés (`bash`, `tcsh`, ...). C'est le shell `bash` qui vous utiliserez cette année.

Nous allons manipuler dans cette séance quelques commandes utiles du shell.

1. dans la suite du document, nous utiliserons le terme unix, linux étant un type d'unix

2 Archivage (niveau ∗)

Il est possible en UNIX d'archiver plusieurs fichiers ou répertoires en les regroupant dans un fichier archive unique. Dans le cas d'un répertoire toute l'arborescence de fichiers dont ce répertoire est la racine est archivée. Cet archivage est utile pour des sauvegardes ou pour communiquer facilement un ensemble de fichiers.

Pour effectuer l'archivage et le désarchivage, on utilise la commande `tar` :

– Archivage : `tar -cvf arch.tar file1 file2 rep1`
regroupe dans le fichier arch.tar les deux fichiers file1 et file2 ainsi que le contenu du répertoire rep1, en préservant l'arborescence.

Attention : Bien spécifier en premier paramètre un nom de fichier d'archivage avec son suffixe .tar, sinon c'est le nom du premier fichier que vous voulez archiver qui sera utilisé comme nom d'archive et donc votre premier fichier sera écrasé et perdu.

– Désarchivage : `tar -xvf arch.tar` extrait le contenu du fichier archive,
– Consultation : `tar -tvf arch.tar` liste le contenu du fichier archive sans extraire.

Faire `man tar` pour plus d'informations.

Travail à effectuer :
– Créer un répertoire `TP3` dans votre répertoire `UtilisationDesOrdinateurs`,
– Récupérer sur la page Utilisation des Ordinateurs, le fichier sujets.tar,
– Dés-archiver cette archive et parcourez-la pour découvrir son contenu,
– Détruire le fichier sujets.tar,
– Reconstruire un fichier d'archive identique à l'original.

Conseil : pour éviter de mal utiliser la commande `tar` et d'écraser par mégarde des fichiers, nous vous conseillons d'archiver des répertoires. Par exemple si vous voulez archiver le répertoire TP3, il suffira de :
– se placer dans le répertoire parent de TP3
– taper `tar -cvf TP3.tar TP3`
Un oubli de `TP3.tar` entraînera une erreur d'utilisation de `tar` mais en aucun cas ne détruira de fichier.

Conseil : Pensez à effacer les fichiers archive une fois que vous avez dé-archivé leur contenu. Dans beaucoup d'enseignements, vous allez récupérer de tels fichiers et, à la longue, vous risquez de ne plus avoir de place mémoire (quota).

3 Gestion des répertoires et des fichiers (niveau ∗)

On rappelle que tout fichier ou répertoire peut être désigné :
– soit par un chemin absolu depuis la racine /
`/bin/ls` est le chemin d'accès au programme qui exécute la commande ls,
– soit par un chemin relatif par rapport au répertoire de travail (*working directory* affiché par la commande `pwd`), aussi appellé répertoire courant. Initialement (en début de session SHELL), le répertoire

travail est le répertoire de connexion. La commande `cd` permet de changer le répertoire de travail (voir TP1) :
`UtilisationDesOrdinateurs/TP3` est un chemin d'accès à partir de votre répertoire de travail.

3.1 Copie, Déplacement, Suppression, Renommage (niveau ∗)

Les commandes `cp` et `mv` utilisent un «argument source» et un «argument destination». Ces arguments sont un chemin d'accès relatif ou absolu vers un fichier ou un répertoire. L'action effectuée dépend de la nature de la source et de la destination :

- `cp f1 f2` : fait une copie du fichier f1 dans un fichier f2 dont le chemin d'accès sera f2. Si un fichier ayant ce chemin d'accès existe, il est écrasé par la copie de f1,
- `cp f1 ... fn rep` : fait une copie des fichiers f1, ..., fn dans le répertoire rep (existant) en préservant le nom des fichiers,
- `mv f1 f2` : déplace le fichier f1 dans un fichier dont le chemin d'accès sera f2. Si un fichier ayant ce chemin d'accès existe il est écrasé par f1,
- `mv f1 ... fn rep` : déplace les fichiers f1 ... fn dans le répertoire existant dont le chemin d'accès est rep. Les fichiers de même nom dans ce répertoire seront écrasés,
- `mv rep1 rep2` :
  - si le répertoire désigné par le chemin d'accès rep2 existe alors le répertoire rep1 et son contenu est transféré dans le répertoire rep2 dont il devient un sous-répertoire,
  - s'il n'existe pas, le répertoire rep1 est renommé en rep2 (changement du chemin d'accès).

Les commandes `rm` et `rmdir` suppriment respectivement des fichiers et des répertoires vides, désignés par leur chemin d'accès :

- `rm f1 ... fn` : supprime les fichiers f1 ... fn,
- `rmdir rep` : supprime le répertoire rep si celui-ci est vide.

3.2 Caractères joker (niveau ∗)

Lorsque le shell interprète les noms de fichiers, il considère certains caractères comme des caractères spéciaux (jokers, aussi appelés *métacaractères*) :

- `*` correspond à un nombre quelconque de caractères,
- `?` correspond à exactement un caractère,
- `[abcdgh]` correspond à exactement un caractère de l'ensemble a,b,c,d,g,h,
- \ indique que le caractère qui suit ne doit pas être considéré comme un métacaractère, ou un caractère spécial du shell(<, >, |) ; cela permet, par exemple de désigner des fichiers dont le nom comporte des métacaractères.

Par exemple :

- `more *.ml` : fait un `more` sur tous les fichiers dont le nom est terminé par .ml,

– `ls *.*` : liste tous les fichiers du répertoire courant dont le nom contient un point,

– `rm a/*` : supprime tous les fichiers du répertoire `a` du répertoire courant dont le nom ne commence pas par un point (i.e. tous les fichiers du répertoire `a` sauf les fichiers cachés),

– `ls TP?/*.c` : liste tous les fichiers dont le nom a pour suffixe `.c`, et qui sont en outre contenus dans des répertoires dont le nom est composé de trois lettres commence par `TP`,

– `ls *?.*?.*` : liste tous les fichiers dont le nom contient au moins un caractère puis un point puis au moins un caractère puis un point puis un nombre quelconque (nul éventuellement) de caractères.

Travail à effectuer :
L'archive que vous avez extraite contient les premiers sujets de TP. On vous demande de réorganiser cet ensemble de répertoires en :
– créant deux nouveaux répertoires ≪Sources≫ et ≪PDF≫,
– transférant dans Sources tous les fichiers .tex et dans PDF tous les fichiers .pdf,
– supprimant les anciens répertoires et les fichiers qui subsistent,
– reconstituant une archive contenant la nouvelle structure.
Vous pouvez recommencer plusieurs fois en utilisant des commandes différentes et en utilisant des chemins relatifs et absolus.

4 Redirections (niveau $**$)

Les calculs en cours (ou processus) échangent des informations avec leur environnement (périphériques, fichiers, ...) sous la forme de suites d'octets (ou flots d'entrées/sorties).

Initialement, le shell dispose de trois flots d'entrée/sortie :
– l'entrée standard (flot numéro 0), associée au clavier,
– la sortie standard (flot numéro 1), associée à l'écran,
– la sortie erreur standard (flot numéro 2), associée aussi à l'écran.
Ces associations des flots standard sont conservées, par défaut pour les commandes lancées à partir du shell. Néanmoins, il est possible de redéfinir ces associations, pour une commande donnée : on parle alors de redirection d'un flot. Ainsi :
– la sortie standard peut être définie à l'aide du caractère `>` :
`ls -l > liste` : permet de récupérer le résultat de `ls` dans le fichier *liste*,

– il est possible de conserver le contenu d'un fichier vers lequel on redirige la sortie standard :
`ls -l >> liste` : ajoute le résultat de `ls` a la fin du fichier *liste*,

– en Bourne Shell (`sh` ou `bash`), la sortie erreur standard peut être redirigée par `2>`
`rm * 2> log` : écrit les messages d'erreurs éventuellement engendrés par l'exécution de `rm` dans le fichier *log*,

– l'entrée standard peut être redirigée par `<`
`sort < tolkien.txt` classe les lignes du fichier tolkien.txt par ordre alphabétique.

5 Tubes (niveau $**$)

La sortie standard d'une commande peut être reliée à (redirigée vers) l'entrée standard d'une autre commande par un *tube* (ou *pipe* en anglais). La seconde commande peut alors utiliser comme point de départ de son traitement les résultats fournis par la première commande. Ce schéma de coordination est appelé un schéma *producteur/consommateur* et sera revu longuement lorsque vous suivrez les cours de Systèmes Centralisés.

L'utilisation d'un tube permet de transmettre un flot de données entre deux commandes sans avoir à passer par un fichier intermédiaire : les données transmises peuvent être conservées dans un tampon, en mémoire vive.

La mise en place d'un tube est définie à l'aide du caractère `|` obtenu en appuyant simultanément sur les touches «Alt Gr» et «-»/«6».

`ls -l | more` : relie la sortie de `ls -l` à l'entrée de `more`. La liste des fichiers du répertoire courant sera traitée par `more`, c'est-à-dire affichée page à page (bien utile quand le nombre de fichiers contenus dans un répertoire est important).

6 Compression (niveau $*$)

Lorsqu'une archive (ou un fichier quelconque) est trop volumineuse, on peut la compresser.

- `gzip rep.tar` : remplace rep.tar par rep.tar.gz compressé,
- `gunzip rep.tar.gz` réalise l'opération inverse.

Si vous désirez produire ou récupérer des archives du monde Windows vous pouvez utiliser les commandes :
- `zip -r9 rep.zip rep` : qui crée l'archive compressée rep.zip à partir du répertoire rep,
- `unzip rep.zip` qui réalise l'opération inverse.

L'option `-r9` sélectionne une compression maximale ; vous pourrez tester l'option `-r1` pour voir la différence.

Travail à effectuer :
- compresser et décompressez l'archive produite précédemment,
- comparer les tailles avec `ls -l`

7 Commandes récursives (niveau $**$)

Les principales commandes de manipulation de l'arborescence des fichiers possèdent une option récursive `-R` qui s'appliquent à un répertoire et à tous ses sous-répertoires :

- `ls -R` : liste récursivement dans un parcours «en largeur» un répertoire et ses sous-répertoires,

- `cp -R rep1 rep2` : fait une copie récursive du répertoire rep1
  - dans un nouveau répertoire de chemin d'accès rep2, si rep2 n'existe pas,
  - dans un sous-répertoire de nom rep1, dans le répertoire rep2 s'il existe déjà !

– `rm -R rep` : détruit tout le répertoire rep et son contenu récursivement ! Utiliser avec prudence ou avec l'option `-i`.

Travail à effectuer :
Expérimenter les options récursives sur l'archive.

8 Alias (niveau ∗)

La commande `alias` permet de d'abréger l'écriture en renommant une fonction ou une expression plus complexe. Par exemple :

```
alias ll="ls -l"
alias del="rm -i"
```

Le deuxième `alias` permet d'utiliser systématiquement la version sécurisée de la commande `rm` qui demande confirmation avant suppression de chaque fichier concerné.

La liste des alias définis est consultable en tapant simplement la commande `alias` sans argument :

```
alias
```

Vous pouvez définir un alias sur une commande en utilisant le nom initial de la commande ; cela modifiera l'effet de la commande :

```
alias mv="mv -i"
alias cp="cp -i"
```

Avec ces alias, lorsque vous déplacerez ou copierez des fichiers/répertoires, il y aura une demande de confirmation en cas de conflit ce qui évitera quelques pertes.

Pour pouvoir utiliser la commande initiale, sans l'effet de l'alias, vous devrez la faire précéder d'un \.

```
\mv fic1 fic2
```

Travail à effectuer : Définir des alias et expérimenter leur effet.

Remarque : la portée d'un alias, c'est à dire l'environnement dans lequel l'alias est défini, est le SHELL courant autrement dit, en simplifiant, le terminal courant. Vous pouvez le constater en manipulant deux terminaux et en vérifiant que les alias de l'un ne sont pas connus de l'autre (à moins bien sûr, de définir les mêmes alias dans les deux terminaux).

On verra à la section 10 comment rendre les alias « permanents ».

9 Variables d'environnement (niveau ∗)

L'interprète de commandes permet de définir des variables d'environnement, c'est-à-dire d'associer des noms à des chaînes de caractères. Cette facilité permet de configurer simplement le comportement de l'interprète de commandes en associant des valeurs particulières à des variables prédéfinies. Par exemple, la variable d'environnement `PWD` contient la valeur du répertoire courant.

Au cours de ce TP nous étudierons les variables locales dont la portée est l'interprète SHELL dans lequel elles ont été définies.

- `env` liste les variables locales et leur valeur,
- `echo $`*variable* : affiche la valeur d'une variable,
- `variable=chaîne` : modifie la valeur d'une variable locale (ou en crée une nouvelle).

Travail à effectuer :

- utiliser la commande `env`, puis la commande `echo` pour consulter les variables déjà définies,
- consulter la valeur de la variable `HISTSIZE`, qui indique le nombre maximum de commandes tapées dans le SHELL mémorisées et accessibles par les flèches haut/bas
- donner lui la valeur 2 et vérifier que le nombre de commandes mémorisées est 2,
- modifier après l'avoir consultée la variable `PS1` (prompt) qui définit l'≪invite≫ de l'interpréteur SHELL :
  - essayez `export PS1="Bonjour :"` ,
  - regardez l'action de `\u`, `\w`, `\W` et `\!` dans la chaîne (en changeant de répertoire pour `\w` et `\W`),
- constater que cette modification n'a pas d'influence sur d'autres occurences de SHELL s'exécutant dans d'autres fenêtres.

10 Fichiers d'initialisation de `bash` (niveau ∗)

Le répertoire de connexion de votre compte contient des fichiers d'initialisation ≪ cachés ≫, dont le nom commence par un point : `.bashrc`, `.bash_login`, `.bash_logout`.

Ils sont listés par la commande

```
ls -a
```

Ils n'existent pas obligatoirement par défaut.

Lors de la connexion d'un utilisateur, l'interprète de commandes exécute les instructions contenues dans le fichier `.bash_login`. Ces instructions ne sont donc exécutées qu'une seule fois.

Les commandes du fichier `.bashrc` sont exécutées chaque fois qu'un interprète SHELL `bash` est lancé (ouverture d'un terminal).

Lors de la déconnexion c'est le fichier `.bash_logout` qui est pris en compte.

Dans ces fichiers, on peut :

- définir des alias,
- modifier des chemins d'accès,
- configurer des programmes,
- ...

Pour prendre en compte les modifications effectuées dans les fichiers de configuration, vous pouvez :

– soit vous déconnecter puis vous reconnecter sur la machine, dans le cas des fichiers `.bash_logout` et `.bash_login`,
– soit lancer un nouveau shell (commande `bash`),
– soit utiliser la commande `source` qui exécute les instructions contenues dans un fichier (`source .bash_login` ou `source .bashrc`).

Travail à effectuer :
– éditer le fichier `.bashrc` pour introduire des alias ou modifier la variable prompt,
– exécuter la commande `source .bashrc` et constater les changements,
– faire la même constatation dans tout nouveau terminal.

11 Deux commandes shell bien utiles (niveau $**$)

`grep [options] exp [f1 ...]`

Affiche toutes les lignes des fichiers f1... contenant l'expression régulière `exp`. Si la commande porte sur plusieurs fichiers, chaque ligne affichée est précédée du nom de fichier la contenant.

Quelques options intéressantes :

- `-i` rend la recherche insensible à la ≪casse" des lettres (minuscules ou MAJUSCULES)
- `-n` indique la ligne où l'expression a été trouvée
- `-l` ne donne que le nom des fichiers où la recherche a été fructueuse (utile par exemple pour éditer les fichiers qui contiennent l'expression recherchée)

```
grep alias $HOME/.bashrc
grep begin *.tex
grep -in The tolkien.txt
```

Remarques :

– il convient, bien entendu, se placer dans un répertoire qui contient les ≪ bons ≫ fichiers,
– la variable globale `HOME` (que nous reverrons au dernier TP) contient le chemin de votre répertoire de connexion.
– `exp` peut contenir des métacaractères permettant de décrire des motifs de recherche. Ces métacaractères sont spécifiques à la commande `grep` et sont différents de ceux du SHELL.

`find repertoires expression...`

Parcourt récursivement les fichiers des répertoires spécifiés et évalue, pour chaque fichier l'expression de gauche à droite. L'expression permet soit de définir des critères de sélection, soit d'effectuer des opérations (affichages, exécution de commandes) sur les fichiers vérifiant les critères de sélection précédents.

```
find . -name '*.txt' -print
```

affiche (option `-print`) tous les fichiers de l'arborescence du répertoire courant (`.`) qui ont comme suffixe `*.txt` (option `-name '*.txt'`).

```
find $HOME -name '*.txt' -print -exec grep -i '^the'  {} \;
```

trouve tous les fichiers de l'arborescence de votre répertoire de connexion ($HOME) dont le suffixe est `*.txt` et pour ces fichiers, affiche leur nom puis, affiche les lignes contenant l'expression `the` en début de ligne (option `-exec` avec la commande `grep -i '^the'`) et s'en va faire le café.

Remarques :

– "{}" désigne le nom des fichiers trouvés précédemment par le `find`,
– le " ;" (précédé de son \) marque la fin du bloc `-exec`,
– dans le dernier exemple, les affichages du `-print` et celui du `grep` sont effectués à la suite l'un de l'autre.

Une dernière utilisation de la commande `find` avec en prime un tube et la commande `xargs` :

```
find $HOME -name '*.txt' -print | xargs grep -i '^the'
```

Cette commande donne le même résultat que la précédente et on se passe des parenthèses et de l'antislash-point virgule. Elle fonctionne comme suit :

– le flux résultat de find est en entrée de la commande `xargs`,
– celle-ci exécute autant de fois la commande en argument (ici `grep`) qu'il y a d'éléments en entrée,
– chacun de ces éléments devenant le dernier argument de la commande `grep`.

12 CQFAR (Ce Qu'il Faut Avoir Retenu)

À la fin de cette séance de TP vous devez :

1. savoir récupérer une archive,
2. savoir archiver et désarchiver le contenu d'un répertoire ou un ensemble de fichiers,
3. savoir copier, déplacer, renommer, supprimer des fichiers et des répertoires,
4. savoir compresser et décompresser un fichier,
5. savoir utiliser les caractères jokers,
6. savoir modifier une variable locale du shell, et créer des `alias`,
7. savoir modifier le fichier `.bashrc` et prendre en compte ces modifications.