Thème 1 – Shell en mode interactif

TP disponible sur la page web personnelle d'Alexandre Sedoglavic : www.lifl.fr/~sedoglav/SHELL/

1.1 Instructions simples

1.1.1 – Comment accéder à l'aide du shell

Exercice 1.1 – less

La commande **less** permet de visualiser le contenu d'un fichier. Elle propose également un certain nombre de commandes permettant de naviguer dans ce fichier, rechercher…
$ **less** /home/enseign/sedoglav/Texte/germinal1.txt

Une liste non-exhaustive d'options utiles de less :
h : aide
q : quitter
/expression : recherche en avant la première occurrence de 'expression' dans le fichier
?expression : recherche en arrière la première occurrence de 'expression' dans le fichier
n : répète en avant la recherche précédente
N : répète en arrière la recherche précédente
z : avance le texte d'une fenêtre
w : recule du texte d'une fenêtre
Remarque : ^ désigne CTRL (indiqué dans l'aide)

Exercice 1.2 – man

La commande **man** permet de visualiser le manuel d'aide de ligne de commande, et même de fonctions utilisées par le système (en C par exemple).
On peut afficher le manuel d'aide de la commande ls ainsi :
$ **man** ls
Cette commande utilise le visualisateur **less** pour afficher l'aide des commandes données en paramètre.
Le visualisateur utilisé par défaut est défini dans la variable d'environnement globale PAGER, que l'on peut afficher de la façon suivante :
$ **echo** $PAGER
On peut forcer man à utiliser un autre visualisateur avec l'option **-P** ou l'option –**pager** :
$ **man -P** more ls
$ **man --pager**=cat mkdir
L'option **-k** de man permet de rechercher l'ensemble des noms de commande contenant une expression. La commande **printf**, utilisée en C pour afficher une chaîne de caractère sur la sortie standard, est très souvent utilisée. Cependant, on peut vouloir afficher une chaîne de caractère sur d'autres sorties (erreur, variable, fichier…). Il existe d'autres fonctions légèrement différentes de printf contenant le nom printf. On peut les afficher via :
$ **man -k** printf

1.1.2 – Manipulation du système de fichiers

Exercice 1.3 – création d'arborescence

A partir de maintenant, ne pas hésiter à abuser de la commande man pour toute commande que l'on veut utiliser.
La commande **wget** permet de télécharger un fichier à partir d'une adresse web.
La commande **mkdir** (make directory) permet de créer un dossier à partir du répertoire courant.
La commande **cd** (change directory) permet de changer de répertoire courant.
La commande **mv** (move) permet de déplacer un fichier.
La commande **ls** (list) permet de lister le contenu du dossier courant (voir son manuel pour plus d'infos sur son usage).
La commande **rm** (remove) permet d'effacer un fichier, il faut ajouter l'option **-r** pour effacer un répertoire. L'option **-f** permet de forcer la suppression.
Le répertoire courant est représenté par **.** , le répertoire parent par **..** .
On peut éviter de réécrire entièrement certaines commandes ou dossiers ou fichiers… en utilisant la complétion (touche tabulation, notée **TAB**). Par exemple, si on tape :
$ **cd** Sh
puis que l'on appuie sur **TAB**, si Shell existe et est le seul document dans le dossier courant à commencer par Sh, le terminal va naturellement afficher :
$ **cd** Shell

Utilisation de ces commandes pour créer l'arborescence de cet exercice (questions 1 et 2) :
$ **wget** www.fil.univ-lille1.fr/~sedoglav/SHELL/Cours01.pdf
$ **mkdir** Shell
$ **cd** Shell/
$ **mkdir** Cours TD TP Personnel
$ **cd** Cours/ **; mkdir** 01
$ **cd** ..
$ **mkdir** TP/01
$ **mv** ../Cours01.pdf Cours/01/

La commande **chmod** donne des autorisations en exécution (x), écriture (r) et lecture (w) aux utilisateurs comme le propriétaire (u comme user), le groupe (g), les autres (o comme others) et tous les utilisateurs (a comme all).
La commande suivante donne les droits en lecture-écriture au propriétaire sur fichier.txt :
$ **chmod u+rw** fichier.txt
La commande suivante supprime les droits en exécution aux autres utilisateurs sur fichier.txt :
$ **chmod u-x** fichier.txt
L'option **-R** permet de gérer les droits sur le contenu entier d'un dossier, par exemple :
$ **chmod -R a+rwx** dossier
Il peut être long de définir les droits spécifiques pour les différents types d'utilisateurs, on utilise des valeurs octales (nombres) pour représenter les différentes combinaisons r, rw, xr, rwx…
Les **valeurs octales des permissions** sont les suivante : **r → 4, w → 2, x → 1**. Il suffit d'additionner les valeurs octales pour déterminer déterminer une permission, par exemple 6 pour rw (4+2).
On représente par trois valeurs octales les permissions pour respectivement le propriétaires, les membres du groupe puis les autres. Ainsi, la commande suivante donne au propriétaire tous les droits (7), au groupe les droits en lecture et exécution (5) et aux autres uniquement en lecture (4).
$ chmod 754 fichier.txt # équivalent à chmod
La commande suivante, très souvent utilisée, donne tout les droits à tout le monde :
$ **chmod 777** fichier.txt

Pour répondre à la question 3, on peut utiliser les 3 commandes suivantes :
$ chmod -R u+rwx Personnel
$ chmod -R g-rwx Personnel
$ chmod -R o-rwx Personnel
Ou plus simplement celle-ci :
$ **chmod -R 700** Personnel

Exercice 1.4 – Liens symboliques

La commande **ls -lrt** (l pour lister, r pour inverser l'affichage, t pour le temps) affiche le contenu du répertoire courant dans l'ordre inverse de date de modification avec différentes informations.
$ **ls -lrt**
La dernière ligne affichée par cette commande dans mon répertoire est la suivante :
drwxrwxr-x 2 alexandre alexandre 4096 août 25 11:04 Cours
Les paramètres affichés sont dans l'ordre les suivants :
1 – la première lettre représente le type de fichier (d : répertoire, - : document, l : lien) et les suivantes les droits pour le propriétaire, les membres de groupes et les autres utilisateurs,
2 – le nombre de "liens physiques de la données" ; le dossier Cours étant vide, il contient deux liens physiques : . (lui-même) et .. (lien vers son dossier parent),
3 – l'identité de l'utilisateur (userid),
4 – le nom de l'utilisateur (user) ; (sur ma machine userid et user sont identiques),
5 – la taille du fichier en octets,
6 – la date (mois + jour + [heure ou année]),
7 – le nom du fichier.

La commande **wc** (word count) affiche le nombre de lignes, de mots et de caractères (ou octets) pour le fichier donné en paramètre. Les options -l, -w, -c permettent de n'afficher que le nombre voulu.

$ **ln -s** /bin/ls foo
$ **ls -lrt**
$ **wc -c** /bin/ls
Le fichier foo contient 7 octets, tandis que /bin/ls en contient 110080, ln -s crée donc un lien symbolique. La commande **ln** sera expliquée dans l'exercice 1.6.

Exercice 1.5 – Liens symboliques

La commande **touch** (sans option) permet de créer un fichier vide.
Voici les commandes permettant de réaliser l'exercice :
$ mkdir villes ; cd villes
$ **touch** Anvers Bruxelles Gand La_Haye Lille Paris Tournai
$ cd .. ; mkdir steden ; cd steden
$ ln -s ../villes/Bruxelles Brussel
$ ln -s ../villes/Gand Gent
$ ln -s ../villes/La_Haye Den_Haag
$ ln -s ../villes/Lille Rijsel
$ ln -s ../villes/Paris Paris
$ ln -s ../villes/Tournai Doornik

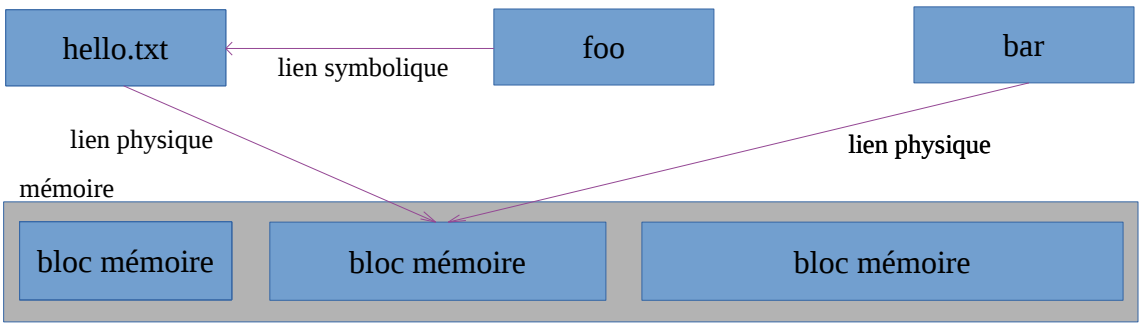
Exercice 1.6 – Liens physiques et symboliques

La commande **cp** permet de copier un fichier ailleurs (usage : cp source destination).

La commande **ln** permet de créer des liens entre des fichiers. La commande **ln -s** permet de créer un lien symbolique vers un fichier, tandis que la commande **ln** sans option crée un lien physique.
Le lien physique accède directement au contenu dans le bloc mémoire, tandis que le lien symbolique n'est qu'une sorte de raccourci vers un autre lien.

Considérons que le répertoire courant ne contient aucun fichier. Nous allons créer un fichier hello.txt ainsi qu'un lien physique et un lien symbolique :
$ echo "Hello !" > hello.txt
$ ln -s foo hello.txt
$ ln bar hello.txt

Ci-dessous un bref schéma de l'accès mémoire des fichiers hello.txt, foo et bar. Le fonctionnement exact sera détaillé dans le cours de Programmation des système.



Effectuons les commandes suivantes :
$ ls > original
$ **ln -s** original symbolic_link
$ **ln** original physical_link
$ **cp** original copy
$ **ls -lrt**
Les fichiers original, copy et physical_link ont la même taille tandis que le fichier symbolic_link est beaucoup plus léger (8 octets).
$ **echo** "une nouvelle ligne" >> original
Après ajout d'une ligne à original, seuls les fichiers original et physical_link sont modifiés.
$ **cat** symbolic_link
Cependant, il y a bien "une nouvelle ligne" lorsqu'on affiche le contenu de symbolic_link.
$ **rm** original ; cat symbolic_link ; cat physical_link
Après suppression de original, symbolic_link n'a plus accès au bloc mémoire de original, tandis que physical_link y a encore accès.
Remarque : ce bloc mémoire ne sera effacé que s'il n'y a plus de lien physique vers celui-ci.

Exercice 1.7 — Création et manipulation d'archive

La commande **tar** permet de stocker et d'extraire des fichiers d'une archive.
Elles dispose notamment des options suivantes (le – n'est pas nécessaire) : **c** (créer l'archive), **v** (active le mode "verbeux" *i.e.* affiche ce qu'il fait), **x** (extract), **f** (utilise le fichier donné en paramètre), **z** (ajoute la compression Gzip)…
Question 1 – archive non-compressée :
$ **tar cvf** Shell.tar Shell
Question 2 – archive compressée (avec Gzip) :
$ **tar cvfz** Shell.tar.gz Shell

Question 3 – lister le contenu d'une archive non-compressée :
$ **tar tvf** Shell.tar fichier.txt
Question 4 – extraction d'un fichier d'une archive non-compressée :
$ **tar xvf** Shell.tar Shell/Cours/01/Cours01.pdf
Question 5 – extraction d'un fichier d'une archive non-compressée :
$ touch foo ; **tar rf** Shell.tar foo
Question 6 – décompresser et ouvrir une archive compressée :
$ **tar xvfz** Shell.tar ; cd Shell

1.1.3 – Manipulation du système de fichiers

Exercice 1.8 – manipulation de processus depuis le shell

La commande **xterm** lance un terminal X.
La commande **emacs** ouvre l'éditeur de texte emacs.
La commande **fg** relance au premier plan la dernière commande ou une commande dont on a le numéro de job.
La commande **bg** relance en arrière-plan la dernière commande.
La commande **kill** permet de supprimer un processus.

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** &
La commande **jobs** montre les commandes en cours d'exécution et terminées ainsi que le numéro du job donné à chaque commande.
$ **jobs**
Si emacs a pour numéro de job 2 alors
$ **fg** 2
Après avoir fait CTRL+Z :
$ jobs
$ **bg** 2
$ jobs
$ **kill** %2

1.1.4 – Redirection

Exercice 1.9 – Éditer avec cat

Sans arguments, la commande **cat** retourne son entrée standard vers sa sortie standard jusqu'à réception du caractère **EOF** (**CTRL+D**).
Les redirections permettent de rediriger ailleurs des affichages. Par exemple la commande ls > original redirigeait l'affichage sur la sortie standard de ls dans le fichier original.
Question 1 – utilisation de redirection pour écrire un fichier :
$ **cat** > /tmp/foo << FINDEPOEME
> Le vierge, le vivace, le bel aujourd'hui
> va-t'il nous …
> FINDEPOEME
La commande précédente > redirige ce qui est entré sur l'entrée standard dans le fichier /tmp/foo qui est soit créé soit écrasé, jusqu'à ce que FINDEPOEME apparaisse.

Question 2 – afficher avec cat :
$ **cat** /tmp/foo
Question 3 – ajout dans un fichier avec une redirection :
$ **cat** >> /tmp/foo << FINDEPOEME
> Prendre par derrière,
> Ce vil coquin !
> FINDEPOEME
La commande << redirige ce qui est entré sur l'entrée standard dans le fichier /tmp/foo sans en écraser le contenu pré-existant.
Question 4 – copie en utilisant les redirections :
On peut copier un fichier dans un autre sans la commande cp. En effet, on peut également afficher le contenu du fichier, et plutôt que de l'afficher sur la sortie standard, on redirige l'affichage vers le second fichier :
$ **cat** < /tmp/foo > /tmp/bar

Exercice 1.10 – Quizz

Question 1 :
La commande suivante stocke dans /tmp/foo l'affichage de la commande ls -al. La ligne "Debut" est écrasée à cause de la deuxième redirection et || (ou) n'est pas réalisé comme l'instruction avant cette dernière a été effectuée :
$ (echo "Debut" > /tmp/foo ; ls -al > /tmp/foo) || echo "Fin" >> /tmp/foo ;
On modifie la commande précédente de la façon suivante afin d'afficher "Debut" et "Fin" respectivement en début et en fin du fichier /tmp/foo :
$ (echo "Debut" > /tmp/foo ; ls -al >> /tmp/foo) && echo "Fin" >> /tmp/foo ;
Question 2 :
Quelque chose avec les filtre page 30 cours me pose problème.

- ls /bin/fichierquinexistepas /bin/ 1 > /tmp/foo1 stocke l'affichage de ls /bin/ dans /tmp/foo1 et affiche sur la sortie standard l'erreur d'accès à /bin/fichierquinexistepas, le rajout d'une redirection envoie ce message d'erreur dans /tmp/foo2. Le 1 et 2 génèrent également des erreurs qui ne sont pas affichées comme il y en a avant celles-ci.
- Le ls de tout ce qui est entre parenthèses est enregistré dans /tmp/foo, 1>&2 ne fait rien. Le 2 n'a pas d'incidence.
- La flemme d'interpréter cette dernière ligne contenant beaucoup de commandes…

Exercice 1.11 – Tube de communications

Chaque tube **|** indique que la commande qui suit le tube agit sur le résultat de la commande qui le précède.
Avec l'option **aux**, la commande **ps** affiche les mêmes informations pour tous les utilisateurs connectés à la machine.
La commande **tail --line=+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* fournit 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 --line=+2 | cut -f1 -d' ' | sort | uniq

1.1.5 – Quelques commandes et précisions utiles

Exercice 1.12 – Historique des commandes

La commande **echo** permet d'afficher sur la sortie standard une chaîne de caractères. On affiche la valeur de la variable HISTSIZE grâce au symbole $:
$ **echo** $HISTSIZE
Mon historique peut mémoriser 1000 commandes.
Une commande interne est un filtre implanté dans le shell et ne correspond (en théorie) à aucun fichier exécutable. La commande history est donc interne.
!ls et **! ls** sont différents car !ls exécute la dernière commande commençant par ls apparaissant dans history tandis que ! ls affiche le résultat de ls sur la sortie standard et réalise la négation du code retour (non affiché à l'écran) de cette commande.

Exercice 1.13 – Expressions régulières et métacaractères

Les commandes répondant à la question 1 sont les suivantes :
$ ls t*
$ ls ??? # affiche quelques entrée à 3 caractères puis d'autres, ne fait pas ce qui est attendu
ls ???*
ls ? ; ls ?? ; ls ???
ls *d*
ls [at]*
ls [r-u]*
Après avoir créé toto, tutu et titi avec "$ touch toto tutu titi", on peut les supprimer en un seul paramètre des deux façons suivantes :
$ rm t[iou]t[iou]
$ rm t?t?

Exercice 1.14 – 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.
La commande **xargs** prend en argument une commande et tout ce qui est lu par xargs sur l'entrée standard est passé comme argument à la commande. Elle se combine très bien avec des | pour effacer des éléments listés par exemple. La dernière question de l'exercice précédent aurait pu se faire ainsi :
$ **ls t?t? | xargs rm**
On peut afficher tous les fichiers de l'arborescence se terminant par un ~ ainsi :
$ **find** . -name "*~" -print
Pour effacer les copies faites par les éditeurs (contenant ~), on peut utiliser :
$ find "*~" -exec rm {} \;
Pour effacer tous les fichiers se terminant par un ~, on peut utiliser un tube de communication :
$ **find** . -name "*~" -print | **xargs** rm

1.2 Variables du shell

Exercice 1.15 – Code de retour

$ ls ; echo $?
Le terminal affiche la liste des éléments du répertoire courant et affiche le code retour de ls : 0.
$! ls ; echo $?
La négation change le code retour de ls. Pareil qu'au-dessus mais avec le code retour 1.
$ (exit 3) ; echo $?
Ce code affiche 3, mais le sens des parenthèses ne m'est pas clair. Exit permet de quitter le terminal sans les parenthèses.

Exercice 1.16 – examen des variables

La commande **set** permet de lister les variables d'environnement et les variables liées au shell.
La commande **grep mot fichier** renvoie les lignes contenant mot dans fichier. L'option **-v** permet d'inverser la sélection (grep -v if n'affichera pas les lignes contenant des if).
Les séries de redirection suivantes permettent d'afficher ce qui est demandé :
$ set | grep = | grep -v if | cut -f1 -d"=" > /tmp/foo
$ set | grep = | grep -v if | cut -f2 -d"=" > /tmp/bar

Exercice 1.17 – la variable PATH

La commande **pwd** permet d'afficher le nom du répertoire courant.
$ echo $PATH
$ cd ; mkdir mesbin ; cd mesbin/ ; ln -s /bin/ls monls
$ pwd
$ PATH=$PATH:'contenu_de_pwd'
$ cd /tmp/ ; monls

Exercice 1.18 – La variable PWD

La variable **PWD** contient le nom du répertoire courant.
La commande PWD=/tmp permet de changer de répertoire.
La variable **HOME** correspond à votre espace personnel et est utilisée par l'interprétation de la commande cd.

Exercice 1.19 – Exportation de variables

Lorsqu'on tente d'afficher FOO dans xterm, rien n'est affiché, sterm ne connaît pas la variable. Il faut l'exporter à partir du processus parent avec la commande **export** pour que FOO soit reconnu.
$ FOO="bar"
$ export FOO
$ xterm
$ echo $FOO

Exercice 1.20 – La variable LANG

La commande externe locale définit la langue utilisée sur le système.
La variable **LC_TYPE** indique comment gérer les caractères et la variable **LANG** indique la langue utilisée. Par exemple, en changent cette dernière variable on modifie les pages de manuel utilisés par **man**.

1.2.1 Mécanisme d'évaluation

Exercice 1.21 – La variable PATH

On peut stocker le premier répertoire apparaissant dans PATH de la façon suivante :
$ HEAD=`echo $PATH | cut -f1 -d':'`
La commande suivante convient également :
$ echo ${PATH %%:*}
Les `` permettent d'affecter le résultat d'une commande. Ainsi la variable **LS=`ls`** contient la liste des éléments du répertoire courant.
On peut stocker les autres répertoires ainsi :
$ TAIL=`echo $PATH | cut -f2- -d':'`

Exercice 1.22 – La variable d'invite

La variable **PS1** du bash est systématiquement utilisée par le shell afin de définir l'invite de commande.
La commande **date** retourne la date.
La variable **USER** contient le login utilisateur.
La variable **HOSTNAME** contient le nom de la machine.
$ PS1='$(date | cut -d" " -f5) ${USER}@${HOSTNAME}: '

1.2.2 Utilisation des variables et du mécanisme d'évaluation

Exercice 1.23 – La commande which

La commande **which** donne l'emplacement d'une autre commande.
La commande which echo nous indique l'emplacement d'echo sur /bin/echo.
Le fichier coucou n'est pas localisé car which recherche les commandes dans les répertoires définis par la variable PATH. Il faut donc ajouter le répertoire /home/enseign/sedoglav dans la variable PATH :
$ PATH=$PATH:/home/enseign/sedoglav

Exercice 1.24 – Les alias

On trouve généralement les **alias** dans le fichier .bashrc. Ils définissent des abréviation à certaines expressions. On peut afficher l'ensemble des alias de .bashrc ainsi :
$ grep 'alias ' .bashrc | grep -v '#'
On définit les alias de l'énoncé ainsi :
$ alias h=history
$ alias lr='ls -lrt'