TP1 - Rappel sur Bash et scripts

A. Rappel sur Shell et principales commandes Unix

(adapté du cours "Introduction au système UNIX"[1] d'E.Viennet)

Shell

Un shell est un interpréteur de commande en mode texte. Il peut s'utiliser en mode interactif ou pour exécuter des programmes écrits dans le langage de programmation du shell (appelés shell scripts).
En mode interactif, le shell affiche une invite en début de ligne (*prompt*), par exemple un caractère $, pour indiquer à l'utilisateur qu'il attend l'entrée d'une commande. La commande est interprétée et exécutée après la frappe de la touche "Entrée". Voici un exemple d'utilisation d'un shell ; les lignes débutants par $, sont entrées par l'utilisateur, les autres sont affichées en réponse :

```
$ pwd
/users/emmanuel/COURS/SYSTEME/POLYUNIX
$ ls
Makefile polyunix.dvi polyunix.tex
fig polyunix.idx polyunix.toc
hello.c polyunix.ind ps
$ ls fig
arbounix.fig tabdesc.fig tube.fig
$ ls -l *.c
-rw-r--r-- 1 emmanuel users 84 Mar 25 1996 hello.c
```

Chaque ligne entrée par l'utilisateur est interprétée par le shell comme une commande, dont il lance l'exécution. Le premier mot de la ligne est le nom de la commande (par exemple `pwd` ou `ls`) ; il est éventuellement suivi d'un certain nombre d'arguments (par exemple fig ou -l).

Metacaractères

Certains caractères, appelés métacaractères, sont interprétés spécialement par le shell avant de lancer la commande entrée par l'utilisateur.
Par exemple, si l'on entre `ls *.c`, le shell remplace l'argument *.c par la liste des fichiers du répertoire courant dont le nom termine par .c.
Les métacaractères permettent donc de spécifier facilement des ensembles de fichiers, sans avoir à rentrer tous leurs noms. Voici les plus utilisés :
– * remplacé par n'importe quelle suite de caractères ;
– ? remplacé par un seul caractère quelconque ;
– [ ] remplacé par l'un des caractères mentionnés entre les crochets. On peut spécifier un intervalle avec - : [a-z] spécifie donc l'ensemble des lettres minuscules.

Exemples :

[1] http://viennet.developpez.com/cours/systeme/unix/

```
$ ls
ABCDEF a grrr prog prog.o
Q.R.S aa hel.l.o prog.c x.y.z
$ ls A*
ABCDEF
$ ls *.c
prog.c
$ ls *g*
grrr prog prog.c prog.o
$ ls *.?
Q.R.S hel.l.o prog.c prog.o x.y.z
$ ls [hg]*
grrr hel.l.o
$ ls *.[a-z].*
hel.l.o x.y.z
```

On peut empêcher l'interprétation des métacaractères par le shell en plaçant l'argument entre apostrophes (exemple : `ls '*'`).

Variables d'environnement

Le système unix définit pour chaque processus une liste de variables d'environnement , qui permettent de définir certains paramètres : répertoires d'installation des utilitaires, type de terminal, etc. Chaque programme peut accéder à ces variables pour obtenir des informations sur la configuration du système.
Depuis le shell *csh*, les variables d'environnement sont manipulées par les commandes `env` (affiche la liste), `setenv` VARIABLE VALEUR (donne une valeur à une variable), et `echo` $VARIABLE (affiche la valeur de la variable).
Depuis le shell *bash*, la liste des variables peut aussi être affichée par `env`, mais pour donner une valeur il faut utiliser `export VARIABLE=VALEUR`, echo peut être utilisé pour afficher la valeur de la variable.

Commandes de manipulation des fichiers

cat [fichier1 ...]

Recopie les fichiers spécifiés l'un après l'autre sur la sortie standard (concaténation). Si aucun argument n'est spécifié, lit sur l'entrée standard (jusqu'à rencontrer un caractère fin de fichier CTRL-d). La sortie standard est normalement l'écran, et l'entrée standard le clavier, donc cat fichier affiche simplement à l'écran le contenu du fichier spécifié.

cd [chemin]

Change le répertoire courant. Sans argument, ramène dans le répertoire de connexion
(HOME).

chmod mode fichier

Modifie les droits d'accès au fichier. Les droits sont spécifiés sous la forme :
ensemble d'utilisateurs +/- type de droit.

Exemples :
`chmod a+r boite` # donne le droit a tous (a) de lire (r) boite;
`chmod og-w boite` # interdit (-) aux autres (o) et au groupe (g) d'ecrire (w);
`chmod u+x boite` # donne le droit d'execution (x) au proprietaire (u) du fichier boite.

cp [-ipr] source... dest

Si dest est un répertoire, copie le ou les fichier(s) source vers dest. Si dest est un nom de fichier, renomme source.
Principales options :
-i demander confirmation en cas d'écrasement de la destination ;
-p préserve les dates d'accès et de modification ;
-r copie récursive.

echo [-n] message

Affiche les arguments, tels qu'il sont évalués par le shell. L'option -n supprime le saut de ligne.

ls [-aldF] chemin1 chemin2 ... cheminn

chemin_i est un nom de fichier ou de répertoire. Si c'est un fichier, affiche sa description ; si c'est un répertoire, affiche la description de son contenu.
Options : -a liste tous les fichiers
(y compris les .* normalement cachés).
-l format long (taille, date, droits, etc).
-d décrit le répertoire et non son contenu.
-F format court avec indication du type de fichier
(ajoute * si exécutable, / si répertoire).
-i affiche les numéros d'inode des fichiers.

mkdir [chemin]

Crée un répertoire. Le chemin peut être relatif (par exemple mkdir ../exam) ou absolu (par ex. mkdir /users/emmanuel/cours).

mv [-i] source dest

Si dest est un répertoire, déplace le fichier source vers dest. Si dest est un nom de fichier, renomme source. L'option -i permet de demander confirmation en cas d'écrasement de la destination.

pwd

Affiche le répertoire courant.

rm [-ri] fichier …

Supprime le ou les fichiers spécifiés. L'option -i permet de demander confirmation pour chacun. L'option -r agit de façon récursive, c'est à dire détruit aussi les répertoires (pleins ou vide) et leurs sous-répertoires.

Redirections des entrées/sorties

Chaque programme sous UNIX dispose au moins de deux flux de données :
l'entrée standard, utilisée en lecture, qui est normalement associée au clavier du terminal, et la sortie standard, utilisée en écriture, normalement associée à l'écran du terminal (ou à la fenêtre de lancement le cas échéant).
Tous les flux de données sont manipulés comme de simples fichiers : on utilisera par exemple la même commande pour lire un caractère au clavier, dans un fichier sur disque ou via une liaison réseau. Ceci simplifie grandement l'écriture des programmes et améliore leur réutilisabilité.

Il est très simple de rediriger l'entrée ou la sortie standard d'un programme lors de son lancement depuis un shell.

Pour la sortie, on utilisera la construction suivante :

```
ls > resultat
```

Dans ce cas, au lieu d'afficher sur l'écran, la commande ls va créer un fichier nommé ici resultat et y écrire. Rien n'apparaît à l'écran (sauf s'il se produit une erreur).

Si l'on désire ne pas effacer l'ancien contenu du fichier, mais écrire à sa fin,
on peut utiliser >> :

```
ls >> resultats
```

Enfin, pour rediriger l'entrée standard, on utilise < :

```
cat < UnFichier
```

Il est possible de rediriger l'entrée et la sortie en même temps :

```
cat < UnFichier > Resultat
```

Notons enfin qu'il existe un troisième flux standard, utilisé pour l'écriture des messages d'erreur (nommé stderr en C, ou cerr en C++). Il se redirige avec >&.

De façon similaire, il est possible de rediriger la sortie standard d'une commande vers l'entrée standard d'une autre commande grâce au tube (pipe) noté |. Les différentes commandes s'exécutent alors en parallèle. Exemple: `ls | sort -r` affiche le contenu du répertoire trié à l'envers.

Contrôle de tâches

Normalement, le shell attend la fin de l'exécution d'une commande avant d'afficher le prompt suivant. Il arrive que l'on désire lancer une commande de longue durée tout en continuant à travailler dans le shell. Pour cela, il suffit de terminer la ligne de commande par le caractère & (et commercial).

Par exemple :

```
$ calcul &
$ ls -Ral / > ls-Ral.txt &
$
```

Ici, les deux commandes s'exécutent en parallèle, tandis que le shell attend notre prochaine instruction. On dit que les processus s'exécutent en tâches de fond. Pour connaître la liste des tâches de fond lancées de ce shell, utiliser la commande jobs :

```
$ jobs
[1]   Running     calcul
[2]   Running     ls -Ral / > ls-Ral.txt
$
```

Le nombre entre crochets est le numéro de la tâche de fond (job). On peut envoyer un signal à une tâche en utilisant kill et %num à la place du PID :

```
$ kill [-signal] %numero_de_tache
```

Pour remettre une tâche de fond au "premier plan", utiliser la commande **fg** (utile si la tâche réalise une entrée clavier par exemple).
Enfin, la commande **sleep n** suspend l'exécution durant n secondes (le processus shell passe à l'état bloqué durant ce temps).

Lorsque l'on entre une ligne de commande sous un shell, ce dernier demande au système d'exécuter la commande, après avoir éventuellement substitué les méta-caractères (*, ? etc.) et remplacés les alias.
A chaque commande doit alors correspondre un fichier exécutable (avec le droit x). Ce fichier exécutable porte le nom de la commande et est recherché par le système dans une liste de répertoires (spécifiée par la variable d'environnement PATH).
La commande which permet de savoir quel est le fichier correspondant à une commande:
which commande

> Affiche le chemin du fichier exécutable correspondant à la commande. Exemple :
> ```
> $ which csh
> /bin/csh
> ```

Le fichier exécutable est soit un fichier binaire (du code en langage machine), qui est alors exécuté directement par le processeur, soit un fichier texte contenant un script.

Scripts

Les scripts sont des programmes écrits dans un langage interprété, par exemple le langage du shell. Un script peut être une simple liste de commandes.
La première ligne du script doit préciser l'interpréteur utilisé, elle commence par les deux caractères **#!**, suivis du chemin absolu de l'interpréteur.

Voici un exemple simple :

```
#!/bin/csh
echo Contenu du repertoire courant
ls -l
echo ---------------------------
```

Ce script affiche la liste détaillée des fichiers du répertoire courant. Il doit être entré à l'aide d'un éditeur de texte, enregistré dans un fichier (nommé par exemple "liste"), puis rendu exécutable grâce à la commande **chmod**:

```
$ chmod a+x liste
```

Exemple d'utilisation du script :

```
$ liste
Contenu du repertoire courant
-rw-r--r--  1 em users  1466 Mar 19 1996 arbounix.fig
-rw-r--r--  1 em users  4066 Feb 17 12:28 filesystem.fig
-rw-r--r--  1 em users  659 Feb 17 13:16 pointpoint.fig
-rw-r--r--  1 em users  583 Mar 19 1996 tabdesc.fig
-rw-r--r--  1 em users  1062 Mar 19 1996 tube.fig
---------------------------
```

On peut écrire des scripts sophistiqués (avec arguments, tests, boucles etc.).

Commandes diverses

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.

head [-n] [fichier]

Affiche les n premières lignes du fichier. Si aucun fichier n'est spécifié, lit sur l'entrée standard.

lpr [-Pnom-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. 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 [-Pnom-imprimante]

Permet de connaître l'état de la file d'attente associée à l'imprimante.

lprm [-Pnom-imprimante] numjob

Retire un fichier en attente d'impression. On doit spécifier le numéro du job, obtenu grâce à la commande lpq.

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).

more [fichier]

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

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.

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.

Filtres

grep [option] motif fichier1 ... fichiern

Affiche chaque ligne des fichiers fichieri 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.

Manipulation des processus

kill -sig PID

Envoie le signal sig au processus de numéro PID. 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.

B. Exercices

1 Shell: remise en forme

(*Interdit d'utiliser gedit ou n'importe quel éditeur de texte pour cette partie*)

1. Écrivez votre nom dans un fichier nom.txt via l'entrée standard (cat).
2. Écrivez votre prénom grâce à echo dans un fichier `prenom.txt`
3. À l'aide des redirections, concaténez les deux fichiers `nom.txt` et `prenom.txt` dans un fichier `id.txt`

4. Quelles sont les différences entre les trois commandes suivantes? Expliquez.

    4.1 `wc toto.c`

    4.2 `wc < toto.c`

    4.3 `cat toto.c | wc`

5. Écrivez une commande renvoyant le nombre d'utilisateurs connectés à votre machine (who, wc).

6. Écrivez une commande pour afficher la liste de tous les fichiers dans votre répertoire ayant:

    6.1 droits de lecture et d'exécution pour le propriétaire, group et tous les utilisateurs

    6.2 droits d'écriture pour le propriétaire seul

7. Changez les droits de votre fichier id.txt pour qu'il soit modifiable par vous uniquement et pour qu'uniquement les membres de votre groupe puissent le lire.

2 Programmation Bash

On peut sauvegarder les commandes complexes et longues dans des fichiers pour éviter de les retaper. Ces fichiers contenant des commandes sont appelés des scripts Shell. Pour exécuter les commandes se trouvant dans un script, il suffit d'entrer le nom du fichier à la ligne de commande. Naturellement, l'exécution d'un script nécessite le droit correspondant (x) sur ce fichier. Généralement la première ligne d'un script indique le type de Shell qui doit l'interpréter et par conséquent la syntaxe suivie pendant son écriture. Les scripts sans indication à la première ligne sont considérés comme des scripts sh (compatible avec bash). Il est préférable d'indiquer systématiquement le type d'interpréteur au début des scripts.
Le Shell peut utiliser des variables locales pour faciliter l'écriture des commandes complexes. En bash, l'affectation à une variable se fait avec '='. La suppression des variables se fait par la commande unset. On obtient la valeur d'une variable en utilisant son nom précédé immédiatement d'un caractère $. Par exemple, le Shell remplacera la chaînes $var1 par la valeur de la variable var1, quand c'est possible.

0. Téléchargez le fichier http://lipn.fr/~buscaldi/systemeTP1.tar.gz et décompressez -le dans un répertoire de votre choix

1. Exécuter le fichier `sc1.sh` donné qui comprend un exemple de « script de bash ». Indiquez les variables utilisées dans ce script. Quelles commandes du script ont défini ces variables, leur ont affecté des valeurs et les ont utilisé ?

Modifier ce script pour répéter la saisie afin d'éviter la chaîne vide à l'entrée. Tester le script modifié.

2. Comme les programmes en C, les scripts acceptent des arguments sur la ligne de commande. Ce mécanisme permet de passer des paramètres aux commandes se trouvant dans le script. Le shell considère ses arguments comme les variables 1, 2, .. et substitue leur valeur aux chaînes $1, $2, … Avec $0 on accède au nom du script et avec $# on obtient le nombre des arguments.
Exécutez le fichier « `sc2.sh` » avec des arguments différents. Expliquer ce que fait ce script. Modifiez sc2.sh avec la commande *shift* pour décaler de 2 la liste des arguments. Vérifiez le comportement du script après cette modification.

2.1. Ecrire un fichier shell qui prend en paramètre 2 entiers et affiche la somme. S'il n'y a pas deux paramètres, il faut afficher un message d'erreur.

2.2. Ecrire un script qui prend en paramètre trois entiers et qui affiche le plus grand. On affichera un message d'erreur s'il n'y pas pas trois arguments passés sur la ligne de commande.

3. En utilisant la structure de contrôle « if … then … else …. » écrire un script bash qui prend en paramètre un nom de fichier et affiche
- un message si le fichier n'existe pas sous le répertoire de travail,
- un avertissement si le fichier est un répertoire, et
- finalement si le fichier est exécutable ou non.

Tester le script écrit en l'exécutant avec plusieurs fichiers de types différentes..

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

4. Taper et sauver le script suivant. Indiquer
- comment on saisie la valeur d'une variable dans un script,
- quel est la signification du cas *),
- le rôle de « ; ; » .

Expliquer le rôle de la variable « essai » en précisant son type.
Programmer les choix 3,4,5.

```
#!/bin/bash
# Exemple de script
clear
essai=1 ;
while [ 0 ] ; do

echo -e " \\n\\n\\n Menu - Essai : $essai "
echo " Afficher repertoire courant 1 "
echo " Lister les fichiers 2 "
echo " Informations sur un fichier 3 "
echo " Changement de repertoire 4 "
echo " n premieres lignes d'un fichier 5 "
echo " Sortie               0 "
```

```
echo -n " Choix : "
read choix ;
echo " "
case $choix in
     1) pwd ;;
     2) ls ;;
     3) ;;      # a remplir ...
     4) ;; # a remplir ...
     5) ;; # a remplir ...
     0) exit ;;
     *) echo Choix non propose ;;
esac
essai=$(( essai + 1 )) ;
done
```

5. On veut écrire un script qui copie les fichiers ordinaires d'un répertoire source vers un répertoire destination. Un fichier du répertoire source n'est copié que s'il n'existe pas de fichier de même nom dans le répertoire de destination. Le script demande d'abord le nom du répertoire source et n'accepte qu'un nom valide (non vide et correspondant à un répertoire existant). Ensuite il demande à l'utilisateur d'entrer le nom du répertoire destination qui doit être également valide. Pour effectuer la copie, il faut vérifier le droit d'écriture sur le répertoire destination. Si ce droit n'existe pas , le script l'ajoute pour le retirer à la fin. Enfin il copie les fichiers.

5.1 Modifier ce script pour qu'il copie chaque fichier du répertoire source vers la destination même s'il y existe un fichier de même nom mais plus ancien. Si la destination contient une version plus récente, la copie n'aura pas lieu. Tester le script modifié.

6. Écrire un script qui concatène puis trie (commande **sort**) deux fichiers `file1` et `file2` dans un nouveau fichier `file3` et qui affiche le nombre total de lignes. Les noms des trois fichiers doivent être passés en paramètre.

7. Dans le but de voir le mécanisme d'exécution des commandes et des scripts, on exécute la commande ps sous le shell et dans deux scripts similaires. Les scripts commencent en affichant leur nom. Comme seconde commande ils affichent la valeur de la variable SHLVL. Ensuite le premier script exécute la commande ps tandis que le seconde exécute le premier script. Tous les deux se terminent avec un dernier affichage qui indique le nom et la fin du script en cours.

A. Ecrire ces deux scripts.
B. Exécuter la commande ps et afficher la valeur de la variable SHLVL dans un shell.
C. Exécuter sous le même shell le premier, puis le second script.

Qu'est ce que l'on peut déduire des affichages obtenus ?

8. On veut écrire un script qui constitue une liste des fichiers ordinaires du répertoire courant, qui répondent à certains critères. Au début du script on définit la liste comme une variable vide. Le script constitue la liste progressivement, en ajoutant à cette variable le nom du fichier que l'on est en train de tester si ce dernier est à sélectionner. Quand on a

terminé le test sur toutes les références du répertoire courant, on affiche la liste à l'écran. Si un argument est donné sur la ligne de commande, il est considéré comme le nom du fichier dans lequel le script doit sauvegarder la liste. Contrairement à l'affichage qui a lieu dans tous les cas, la sauvegarde est optionnelle.
Ecrire le script décrit ci-dessus pour sélectionner les fichiers sur lesquels le propriétaire a le droit d'écriture. Tester le script dans un répertoire qui contient au moins 3 fichiers à sélectionner, un sous-répertoire et plusieurs fichiers à exclure. N'oubliez pas de tester l'option.

8.1. Ecrire un script qui prend en paramètre un entier et affiche l'entier en ordre inverse (123 -> 321).

8.2. Ecrire un script qui prend en paramètre un entier et affiche la somme des digits qui le compose (123 -> 1+2+3 = 6).

9. Expliquer ce qui est fait par la fonction checkpid( ) suivante:

```
checkpid() {
        local i

        for i in $* ; do
                [ -d "/proc/$i" ] || return 1
        done
        return 0
}
```

Une fonction définie peut être utilisée comme une commande.
Utiliser la fonction checkpid( ) pour observer au moins une fois chaque valeur qu'elle peut retourner.

10. (facultatif) Téléchargez le fichier http://lipn.fr/~buscaldi/space.sh . Étudiez le code de ce script. Listez les fonctions définies dans ce script. Modifiez les touches nécessaires pour déplacer le vaisseau et tirer (au lieu de a,l,f utiliser des autres touches de votre choix).