Université Grenoble Alpes
U.F.R. Sciences de l'Homme et de la Société
L3 MIASHS

TP1 : Langage de commande Shell d'UNIX

Cette première partie est un entraînement aux manipulations élémentaires du shell et des fichiers Unix. Vous devez simplement effectuer le travail demandé et poser toutes les questions que vous jugerez utiles. L'objectif est que vous soyez capable de refaire seul(e) les manipulations décrites.

1 Informations sur le système utilisé

– Ouvrez une fenêtre Terminal

– Déterminez le système installé par la commande `uname -a`

– Affichez la distribution utilisée : `cat /etc/issue`

– Aide pour une commande : `nomdelacommande --help`

– Manuel d'une commande : testez la commande `man` pour obtenir de l'aide sur `man`.

2 Manipulation de fichiers et répertoires

– Copiez le répertoire `~adamj/shell` dans votre répertoire de travail par défaut par la commande suivante :

```
cp -r ~adamj/shell .
```

Placez-vous dans votre répertoire **`shell`** et inspectez son contenu.

– Les fichiers du répertoire `noms` sont vides, utilisez la commande `ls -l` pour le vérifier.

– Placez-vous dans le répertoire `noms` (commande cd).

– Faites afficher tous les fichiers dont le nom se termine par .c

– Afficher le nom de tous les fichiers dont le nom commence par un a ou un g.

– Faites afficher le nom de tous les fichiers dont le nom contient exactement 3 caractères.

– Sans changer de répertoire, trouvez au moins deux méthodes pour afficher le contenu de votre répertoire de travail par défaut (HOME directory). Affichez tous les fichiers y compris les fichiers cachés du répertoire de travail.

– Remontez au répertoire `shell`.

3 Compilation et arrêt de programme

Vous êtes dans le répertoire `shell`.

– Affichez le contenu du fichier `bravo.c` par la commande `cat bravo.c`

– Affichez le contenu du fichier du fichier `bravo.c` par la commande `od -c bravo.c` que fait cette commande ?

– Affichez le contenu du fichier du fichier `bravo.c` par la commande `od -x bravo.c` que fait cette commande ?

– Compilez puis lancez l'exécution du programme C contenu dans le fichier `bravo.c` ; pour compiler utilisez la commande cc : `cc bravo.c`
Cette commande crée un fichier exécutable de nom `a.out`

La commande `a.out` lance l'exécution du programme objet ; l'exécution échoue, car **.** (le répertoire courant) n'est pas présent dans la variable $PATH ; pour le vérifier affichez la variable PATH : `echo $PATH`

Il faut donc lancer l'exécution du programme objet par `./a.out`

– Recommencez mais en utilisant l'option `-o` qui permet de redéfinir le nom du fichier contenant l'exécutable du programme (que vous appellerez bravo).
`cc bravo.c -o bravo` produit un fichier exécutable de nom **`bravo`**
`./bravo` lance l'exécution du programme objet.

– De la même manière, compilez le programme `boucle.c` dans le fichier `boucle` et lancez son exécution. Pour l'arrêter, appuyez simultanément sur les touches Ctrl et C du clavier.

3 Redirigeons

Vous êtes toujours dans le répertoire shell et vous avez créé un fichier exécutable de nom `bravo`.
– Lancez l'exécution de bravo en redirigeant sa sortie standard dans un fichier nommé `out`.
– Recommencez mais en AJOUTANT la sortie de bravo à `out` (>> au lieu de >).
– Tapez `ls titi` en redirigeant la sortie erreur de ls sur un fichier `err`.
– Tapez `ls titi *` avec erreurs redirigées. Recommencez mais cette fois en redirigeant la sortie standard. Observez la différence : il s'agit de comprendre qu'un processus Unix a bien deux sorties : une pour les sorties normales et une pour les messages d'erreur (comme Windows).
La commande `cat` envoie le contenu des fichiers passés en paramètre sur sa sortie standard. Si aucun fichier n'est passé en paramètre, c'est le contenu de son entrée standard qui est envoyé sur la sortie.
Vous êtes toujours dans le répertoire shell.
– Utilisez `cat` pour faire afficher à l'écran le contenu des fichiers `m1` et `m2` en une seule commande.
– Utilisez cat pour créer une copie du fichier `m1` baptisée `m1bis`.
– Utilisez cat pour créer un fichier `toto` contenant un texte que vous taperez au clavier (terminer le texte par Ctrl-D qui est la marque de fin de fichier sur Unix).

4 Prise en main de l'environnement

– Revenez à votre répertoire de travail par défaut (HOME directory), il existe au moins 3 syntaxes différentes de la commande cd pour faire cela.
– Faites afficher votre environnement avec la commande **`env`**, puis avec la commande **`set`**. Observez les différences. La variable **PS1** est-elle une variable publique ?
– Le prompt est défini par la variable de l'environnement **PS1**. Modifiez cette variable.
– Ajoutez à la fin de votre fichier **`.bashrc`** la commande `echo -e "\nBonjour\n"`. Ouvrez une nouvelle fenêtre Terminal, vous devez voir s'afficher `Bonjour`.

5 Faisons le ménage

– Sans changer de répertoire (vous êtes dans votre répertoire de travail par défaut), supprimez tous les fichiers du répertoire shell/noms.
– Supprimez le répertoire shell/noms.
– En une seule commande, supprimez le répertoire shell et son contenu.

Dans cette seconde partie, nous allons créer un certain nombre de scripts shell (il s'agit de fichiers texte contenant des commandes du shell). La plupart pourront vous être utiles par la suite, nous commençons donc par organiser l'environnement de travail de façon à pouvoir classer les scripts et les utiliser comme de nouvelles commandes du système.

6 Création d'un répertoire pour ranger vos scripts

Avec la commande `mkdir`, créez un répertoire `bin` dans votre répertoire de travail par défaut (peut-être existe-t-il déjà ?). Positionnez-vous dans `bin` et créez à l'aide de l'éditeur SciTE un fichier `essai` contenant un script shell qui affiche ses deux premiers paramètres sur 2 lignes séparées.

Exemple :
`essai un deux` doit afficher :
```
param 1 : un
param 2 : deux
```

Rendez-ce fichier exécutable (`chmod +x essai`) et lancez-le (`essai un deux`).
Si ce n'est pas déjà fait, il faut maintenant ajouter le répertoire bin que vous venez de créer dans la liste des répertoires où le système va chercher les commandes (PATH), et ce de manière permanente.
Pour ce faire :
– éditez votre fichier `.bashrc` et modifiez-le si nécessaire ;
– ouvrez une nouvelle fenêtre terminal ;
– lancer essai dans cette fenêtre pour vérifier que ça fonctionne.

Pourquoi cela ne marche-t-il pas dans l'ancienne fenêtre ?
Vous disposez maintenant d'un répertoire pour ranger vos commandes personnelles, répertoire qui vous est accessible quel que soit votre répertoire de travail.

7 Substitutions, vous avez dit substitutions ?

Essayez la commande essai *. Expliquez ce qui se passe.
Essayez la commande date.
Essayez la commande essai `date`. Expliquez ce qui se passe.

8 Combien d'utilisateurs ?

La commande **`who`** affiche la liste des utilisateurs connectés (1 par ligne). La commande **`wc`** compte les lignes, les mots et les caractères contenus dans les fichiers passés en paramètres ou sur son entrée standard si aucun fichier n'est passé.
Essayez `who`. Réessayez `who` en redirigeant sa sortie sur l'entrée de `wc` (avec un tube).
En utilisant who en conjonction avec wc (faire man wc pour déterminer la bonne option), créer dans votre répertoire `bin` une commande `hmu` qui affiche :
```
Il y a XX utilisateurs connectés.
```
XX est bien sûr le nombre d'utilisateurs effectivement connectés.
Comme vous êtes seuls connectés sur le poste de travail, cette commande présente peu d'intérêt.
Connectez-vous au serveur imss-dc avec la commande :
`ssh imss-dc` puis votre mot de passe et re-testez `who` et votre commande `hmu`.

9 Soyons prudents...

Unix est parfois un peu violent et ne donne pas droit à l'erreur. Ainsi, un fichier détruit par la commande **`rm`** ne peut pas être récupéré. Pour éviter ça, on va écrire une commande **`del`** qui envoie les fichiers passés en paramètres dans un répertoire (baptisé Poubelle) de votre répertoire de travail par défaut (HOME directory). On utilisera `del` pour effacer des fichiers, ainsi en cas de fausse manœuvre, on pourra récupérer les fichiers dans le répertoire Poubelle. De temps à autre, on ira vider la poubelle avec la commande `rm`.

Revenez au répertoire de travail par défaut et créez le répertoire Poubelle.
Créez un fichier `del` dans votre répertoire `bin`. Ce fichier devra contenir un script permettant de déplacer (commande `mv`) tous les fichiers passés en paramètres dans le répertoire Poubelle.
Vous prendrez garde à ce que ce script fonctionne bien quel que soit le répertoire de travail au moment de l'appel.

Rendez le fichier `del` exécutable et testez-le, en commençant par des fichiers bidons, créés pour l'occasion (on peut créer un fichier vide avec la commande `touch` : **`touch toto titi`** créera les fichiers toto et titi).

10 Compilation paramétrée

Pendant une session de travail sous Unix, on effectue souvent un cycle édition-compilation-exécution et on est donc amené à taper successivement des commandes du type :

```
SciTE tp.c &
```

puis de manière répétitive :

```
cc tp.c -o tp
tp
```

Comme il y a (parfois) des erreurs de compilation, on redirige la sortie des erreurs du compilateur sur un fichier qu'on peut ensuite consulter avec more.

On se propose ici d'automatiser quelque peu ce processus répétitif.

Créez (toujours dans votre répertoire `bin`), un fichier **`comp`** qui compile un programme C en redirigeant la sortie erreur dans un fichier **`comp.log`**. Rendez ce fichier exécutable et testez-le (par exemple sur le programme **`bravo.c`** récupéré dans la première partie.

Modifiez le fichier `comp` :
– pour qu'il produise un programme exécutable de même nom que le paramètre (on devra supposer que **ce paramètre ne contient pas l'extension .c**).
– pour qu'il teste le résultat de la commande `cc` et soit lance l'exécution (en cas de succès), soit affiche `comp.log` en utilisant la commande more. Ainsi, l'appel **`comp bravo`** doit compiler le fichier `bravo.c` et le lancer si la compilation réussit ou bien produire un fichier `comp.log` et l'afficher si la compilation échoue.

11 Processus

Lancez la commande **`comp`** de l'exercice précédent en arrière-plan puis immédiatement la commande **`ps`** (qui affiche la liste de vos processus actifs).

Lancez ensuite la commande ps -ef pour avoir la liste de tous les processus actifs du système.
Pour voir le début de la liste, créez un tube de ps -ef sur more : `ps –ef | more`
Quel est le numéro (PID) du processus init ?
Créez une nouvelle fenêtre avec la commande :

`xterm -T Nouveau &` le & permet de garder la main dans le terminal d'origine

Tapez la commande **`ps -fU $LOGNAME`** pour avoir la liste de tous vos processus en format long.

Notez le numéro (PID) du processus xterm que vous venez de créer. Quel est le numéro de son père (PPID) ? Ce processus a-t-il d'autres fils ?

Avec la commande kill, tuez (arrêtez) le processus xterm (Nouveau) :

`kill -KILL <PIDduProcessus>`

Que se passe-t-il ?

12 Manipulation de plusieurs fichiers

Ecrire un script shell (à ranger dans $HOME/bin) qui à la fin de chacun des fichiers dont le nom est passé en paramètre (supposés être des sources C, C++ ou Java) ajoute la ligne :
/* ------- Fin de <NomDuFichier> ------- */