R 1.04

2023 – 2024

Introduction aux systèmes d'exploitation et à leur fonctionnement

TD N°8

« Programmation en Shell 2 »

ANNE Jean-François
D'après le TD de F. BOURDON

Le but de ce TD est de se familiariser avec les systèmes d'exploitation et avec leur fonctionnement.

« Programmation en Shell 2 »

Notions vues dans ce TD :

La programmation en Shell (variables de position et autres variables).

Nombre de séance de 2h00 prévu pour faire ce TD : 1.

PS : Les parties correspondant à du travail à faire sont toutes en italiques ; le restant étant du complément au cours.

L'instruction « shift » permet de décaler les paramètres de position ($0, $1, $2, ...), sauf le premier qui ne bouge pas. Ce dernier ($0) correspond au nom de la commande en cours.

✍ *Q1. Ecrire une première version d'un script (mon_script1) qui affiche la valeur des paramètres de position, jusqu'à « $10 » compris, passés à l'appel de ce script. Vous pouvez utiliser l'instruction « if [-n "$i"] ; then … ; fi ».*

✍ *Q2. Cette première version ne permet pas d'afficher un nombre variable de paramètres passés à l'appel du script. Proposer une deuxième version (mon_script2) qui le permet. Vous utiliserez l'instruction « while [-n "$i"] ; do … done » et la commande « shift ».*

✍ *Q3. Que se passe-t-il si vous passez les arguments suivants aux scripts « mon_script1 » et « mon_script2 » ? Qu'en concluez-vous ? Les deux guillemets sont passés comme paramètres sans caractère blanc entre eux.*

```
prompt> mon_script1 un "" deux trois
 ???
prompt> mon_script2 un deux "" trois quatre
 ??
prompt>
```

✍ *Q4. Pour corriger le problème vu à la question précédente, modifier « mon_script2 » en « mon_script3 » en utilisant le paramètre « $# ». Afficher la valeur de la variable « $# » dans un script auquel vous passerez des paramètres, pour comprendre ce qu'elle contient.*

Pour la fin de cette question Q4. (lignes suivantes) vous devez tester les consignes afin de bien les comprendre en les mettant en pratique. N'hésitez pas à tester ces consignes sur d'autres exemples.

Le paramètre « $@ » permet de récupérer l'ensemble des arguments passés à l'appel d'un script. Par exemple la commande suivante affiche les informations sur les fichiers passés en paramètre :

```
prompt> cat ll.sh
#! /bin/bash
ls -l "$@"
prompt> ll *.c
 ???
prompt>
```

La protection d'un caractère afin qu'il ne soit pas interprété par le shell, peut se faire soit par un « \ » (barre oblique inverse - backslash), soit par des apostrophes « ' ' », soit encore par des guillemets « " " ».

Le backslash protège uniquement le caractère suivant, y compris lui-même.

```
prompt> echo \$a
???
prompt> echo Fa \# ou Do \#
???
prompt> echo \*\*\*
???
prompt> echo un \\ précède \$5
???
prompt> echo début \
> et fin
???
prompt>
```

Les apostrophes protègent toute une expression en une seule fois. Un backslash devient un caractère comme un autre qui ne peut protéger une apostrophe. L'apostrophe ne peut pas être protégée entre deux apostrophes.

```
prompt> echo '#\$"&>|'
???
prompt>
```

Les guillemets permettent de garder protéger tous les caractères spéciaux sauf « $ », « ' » et « \ », qui conservent leur signification particulière. Les guillemets permettent aussi de garder l'unité d'une chaîne sans la fragmenter en différents mots. Enfin ils préservent les espaces, les tabulations et les retours chariots contenus dans une expression.

```
prompt> var=$(ls /dev/sda1*)
prompt> echo $var
???
prompt> echo "$var"
???
prompt>
```

On peut manipuler des tableaux en Shell. Comme en « C » les index des tableaux sont numérotés à partir de 0.

tableau[i]=valeur # affectation

${tableau[i]} # consultation
${tableau[@]} # la liste de tous les membres du tableau
${#tableau[i]} # longueur du ième membre du tableau
${#tableau[@]} # nombre des membres du tableau

```
prompt> var="valeur originale"
prompt> echo $var
???
prompt> echo ${var[0]}
???
prompt> var[1]="nouveau membre"
prompt> echo ${var[0]}
???
prompt> echo ${var[1]}
???
prompt> echo ${#var[@]}
???
prompt> echo ${#var[0]}
???
prompt> echo ${#var[1]}
???
prompt>
```

On peut évaluer une expression avec la commande « eval ».

✍ *Q5. En vous aidant du cours (« while », « read » et « if […]), écrivez ce script et lancez-le. Que fait-il ? Vous pourrez le tester avec les instructions suivant le script ou d'autres à votre choix.*

```
prompt> cat script
#! /bin/bash
while true ; do
echo -n "? "
read ligne
if [ -z "$ligne" ] ; then
break;
fi
eval $ligne
done
prompt>

prompt> script
? ls
???
? A=123456
? echo $A
???
? B='A = $A'
? echo $B
???
? eval echo $B
???
 ? (entrée)
```

```
prompt>
```

✍ *Q6. Ecrire un script « mes_carres » qui affiche les carrés des différentes valeurs entières passées en paramètre. Vous utiliserez la structure de contrôle « for ».*

```
prompt> mes_carres 1 2 3
1² = 1
2² = 4
3² = 9
prompt>
```

✍ *Q7. Ecrire une fonction « ma_somme » qui affiche les sommes des différentes valeurs entières passées en paramètre. Vous utiliserez la structure de contrôle « for ».*

```
prompt> ma_somme 1 2 3
6
prompt>
```

✍ *Q8. Ecrire une fonction « gco » qui prend en paramètre un fichier source « C » (par exemple « mon_prog.c ») et qui appelle le compilateur C (cc ou gcc) avec l'option " -o nom-source " (dans l'exemple « mon_prog »). On pourra mettre cette fonction dans le fichier " $HOME/.bashrc ". Vous utiliserez la fonction « basename ».*

```
prompt> gco mon_prog.c
prompt> ls
mon_prog.c mon_prog
prompt>
```

✍ *Q9. Ecrire la procédure « fdate.sh » qui affiche la date en français. Vous utiliserez la commande « set », l'instruction « $(cmde) » et la structure « case ».*

```
prompt> chmod u+x fdate
prompt> fdate
Lundi 8 fevrier 2099 20:29:10
prompt>
```

✍ *Q10. Ecrire un script (archiv.sh) qui renomme tous les fichiers du répertoire courant dont l'extension est « .tgz », en fichiers « .tar.gz ». Vous utiliserez la structure de contrôle « for » et l'instruction « ${%}.*

```
prompt> ls *.tgz
a.tgz b.tgz
prompt> archiv
prompt> ls *.tar.gz
a.tar.gz b.tar.g
prompt>
```

✍ *Q11. Ecrire la procédure « new_suff.sh » qui permet de modifier le suffixe de un ou plusieurs fichiers. Le premier paramètre décrit l'ancien suffixe, le second décrit le nouveau et le troisième les fichiers touchés par cette modification. Vous utiliserez les structures « case » et « for », la fonction « shift », les fonctions « mv » et « basename ».*

❖ ***Exemple :***

```
prompt> ls
f1.c f2.c f3.c g1.c
prompt> new_suff .c .old_c f*.c
prompt> ls
f1.old_c f2.old_c f3.old_c g1.c
prompt>
```

✍ *Q12. Ecrire la procédure « comp » qui permet de compiler un fichier. Cette procédure doit contrôler l'existence du fichier source, appeler le bon compilateur (on utilisera l'extension « .c » pour le compilateur C, cc et « .cc » pour le compilateur C++, gcc) et créer un exécutable dont le nom est celui du fichier source sans extension.*

---

1. Créer des commandes personnalisées

a) Première approche : les alias

Placez-vous dans un répertoire dans lequel vous aurez créé quelques fichiers. Vous devez compléter la ou les lignes marquées par les trois points d'interrogation (???).

- ***Généralités***

```
$ alias
alias cd..='cd ..'
alias l='ls'
alias ll='ls -l'
alias ls='ls -F --color=auto'
alias md='mkdir'
alias rd='rmdir'
alias rm='rm -i'
alias s='cd ..'
$ alias ll
???
$ ll
???
$ ls -l
???
```

- ***Créer et supprimer un alias***

```
$ alias lald='ls -ald'
$ alias
???
$ lald
???
$ ls –ald
???
$ alias lald
???
$ unalias lald
$ alias lald
???
$ alias
???
```

- ***Redéfinir une commande ou un alias***

A l'aide de la commande man, déterminer le rôle de l'alias ls. Toujours à partir du même répertoire que précédemment, compléter le ou les lignes marquées par les trois points d'interrogation (???).

```
$ alias ls='ls --color=always'
$ ls
???
$ ll
???
$ unalias ls
$ ls
???
$ ll
???
```

Créer un nouvel alias ls qui permet d'affecter une couleur par type de fichier et vérifier que le nouvel alias affecte tous les autres alias utilisant ls.

b) Seconde approche : les scripts SHELL

✍ *Créer un répertoire ~/bin.*

✍ *Créer à l'aide d'un éditeur un fichier script shell lald dans ce répertoire.*

✍ *L'exécuter ... Que se passe-t-il ?*

✍ *Comment faire pour l'exécuter.*

```
???
```

✍ *Comment faire pour pouvoir l'exécuter de n'importe où ... en utilisant export et la variable PATH*

```
export ?
```

✍ *Comment pérenniser cette propriété ?*

✍ *Ajouter la ligne de commande « echo Chargement du fichier .bash_profile » au début du fichier .bash_profile*

```
# .bash_profile

echo ???

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
. ~/.bashrc
fi

export BASH_ENV=$HOME/.bashrc
```

✍ *Ajouter la ligne de commande « echo Chargement du fichier .bashrc » au début du fichier .bashrc*

```
# .bashrc

echo ???

# Source global definitions
if [ -r /etc/bashrc ]; then
. /etc/bashrc
fi
```

✍ *Logger vous sur un terminal virtuel (CTRL+ALT+F1) ... Que se passe-t-il ?*

✍ *Retourner sous l'environnement graphique et lancer une nouvel console ... Que se passe-t-il ?*

✍ *Comment faire pour que la commande lald soit toujours et automatiquement exécutable de n'importe où ?*

```
# .bashrc

echo Chargement du fichier .bashrc

# Source global definitions
if [ -r /etc/bashrc ]; then
. /etc/bashrc
fi

export PATH=???
```

✍ *Retourner dans une console déjà ouverte et lancer lald. Que se passe-t-il ?*

```
$ lald
bash: lald: command not found
```

✍ *Ouvrir une nouvelle console et lancer lald. Que se passe-t-il ?*

✍ *Expliquer pourquoi.*

```
Chargement du fichier .bashrc
$ lald
drwxr-xr-x 9 meric meric 4096 Sep 16 16:40 .
```

✍ *Comment faire pour que dans la console déjà ouverte on puisse lancer lald ? Vous utiliserez la commande « . » ou « source »*

```
$ ???
Chargement du fichier .bashrc
[meric@localhost meric]$
```

2. Projet

Vous êtes root et vous vous apprêtez à arrêter le système. Par précaution, vous voulez prévenir tous les utilisateurs connectés. Ecrire un SHELL qui automatise cette tâche.

Demander à votre binôme de se connecter sur un ou plusieurs terminaux. CRTL + ATL + Fi, ou « su login » dans plusieurs fenêtres shell.

✍ *Comment connaître la liste des utilisateurs connectés ?*

```
$ who
???
```

✍ *Comment leur envoyer un message ?*

```
$ echo message | write user2
```

ou

```
$ echo Message > /dev/tty1
```

✍ *Que faut-il (en terme de droits) pour pouvoir envoyer ce message ?*

```
???
```

✍ *Comment faire pour le savoir ?*

```
???
```

✍ *Comment modifier les droits ?*

```
???
```

Pour savoir si le terminal, sur lequel un utilisateur est connecté, accepte l'écrire de messages, il suffit de voir la présence du signe « + » en deuxième colonne.

```
$ who -T
meric + tty1 Sep 16 16:36
user2 + tty2 Sep 16 15:29
root + tty3 Sep 16 15:29
user1 + tty4 Sep 16 15:29
meric ? :0 Sep 16 15:15
meric + pts/0 Sep 16 15:15
```

La boucle for ...

✍ *Créer un shell déroulant le code suivant :*

```
#!/bin/sh

for i in `who -T`
do
echo $i
done
```

✍ *Ecrire une commande qui permet de récupérer la liste des utilisateurs auxquels vous pourrez envoyer un message. utiliser les commandes who et grep*

```
$ ???
meric + tty1 Sep 16 16:36
user2 + tty2 Sep 16 15:29
root + tty3 Sep 16 15:29
user1 + tty4 Sep 16 15:29
meric + pts/0 Sep 16 15:15
```

✍ *Demander aux utilisateurs de vous accorder les droits d'écriture.*

```
$ mesg y
```

✍ *Ecrire la commande qui permet de récupérer le nom du terminal. Vous partirez du résultat de la commande précédente que vous enverrez par un pipe dans la commande « awk » :*

```
$ ??? | awk '{ print $3 }'
tty1
tty2
tty3
tty4
pts/0
```

✍ *Créer un shell déroulant le code suivant, après l'avoir complété avec le résultat de la commande précédente :*

```
#!/bin/sh

for i in `??? | awk '{ print "/dev/" $3 }'`
do
echo $i
done
```

✍ *Exécuter la commande.*

✍ *Enrichir le shell pour envoyer le message "Le système va être arrêté !!!" à chacun des terminaux.*

```
#!/bin/sh

for i in `??? | awk '{ print "/dev/" $3 }'`
do
echo > $i
echo "****************************************" > $i
echo ???
echo "****************************************" > $i
done
```

✍ *Enrichir le shell de façon à préciser quand le système sera arrêté. La durée en seconde sera passée en paramètre ($1).*

```
#!/bin/sh

for i in `??? | awk '{ print "/dev/" $3 }'`
do
echo > $i
echo "****************************************" > $i
echo ???
echo "****************************************" > $i
done
```

✍ *Quel est la commande qui permet de connaître le propriétaire d'un terminal (/dev/tty2) ? Vous utiliserez la commande « awk ».*

```
$ ls -l /dev/tty2 | ???
user2
```

✍ *Enrichir le shell de façon à personnaliser le message de la façon suivante :*

```
************************************
Attention meric !!!
Le système va être arrêté dans minutes
************************************
```

```
#!/bin/sh

for i in `??? | awk '{ print "/dev/" $3 }'`
do
echo > $i
echo "****************************************" > $i
echo Attention `???` !!! > $i
echo Le système va être arrêté dans ??? minutes > $i
echo "****************************************" > $i
done
```

✍ *Comment faire pour garder une trace des messages transmis dans un fichier.*

✍ *Donner une solution consistant à spécifier le nom du fichier en paramètre.*

```
#!/bin/sh

echo >> $2
date >> $2

for i in `??? | awk '{ print "/dev/" $3 }'`
do
echo > $i
echo "****************************************" > $i
echo Attention `???` !!! > $i
echo Le système va être arrêté dans ??? minutes > $i
echo "****************************************" > $i
echo `ls -l ??? | awk '{ print $3 }'` "("???")" est averti >> ???
done
```

3. ***Projet***

✍ *Créer un shell qui compte le nombre de sous-répertoire du répertoire courant, qui liste le nom de ces sous-répertoires et qui pour chacun d'eux, affiche le nombre de leurs sous-répertoires.*

```
$ nbsousrep trace
****************** Sun Sep 16 17:38:41 GMT 2001 ********************
Nombre total de répertoires : 2
Nombre de sous-répertoires de [.]: 0
```

```
Nombre de sous-répertoires de [..]: 7
********** Sun Sep 16 17:38:41 GMT 2001 **********
$ more trace
drwxr-xr-x 2 meric meric 4096 Sep 16 17:38 .
drwxr-xr-x 9 meric meric 4096 Sep 16 16:40 ..
```

✍ *Créer un shell qui affiche le nombre de répertoires contenu dans le répertoire courant :*

```
#!/bin/sh

echo
echo "**********" `date` "**********"
ls -la | grep "^d" | tee $1 | wc -l
echo "**********" `date` "**********"
echo
```

```
#!/bin/sh

echo
echo "**********" `date` "**********"
echo "Nombre total de répertoires : " `ls -la | grep "^d" | tee $1 | wc -l`
echo "**********" `date` "**********"
echo
#!/bin/sh
```

✍ *Comment récupérer le nom des sous-répertoires. Vous utiliserez un pipe et la commande « awk » avec la commande faite dans le script précédent.*

```
$ ???
```

✍ *Comment récupérer le nombre de liens hard des sous-répertoires :*

```
$ ???
```

✍ *Déterminer le nombre de sous-répertoires des sous-répertoires :*

```
$ ??? { print $2 - 2 }'
```

II. Webographie :

- https://bourdon.users.info.unicaen.fr/cours/IUT-1A/index.html#exercices
-