TP 03 : Shell bash

Vous rédigerez un compte rendu, sur lequel vous indiquerez la réponse à chaque question ou points clefs, vos explications et commentaires (interprétation du résultat), et le cas échéant la ou les commandes utilisées.

Le compte rendu sera à rendre au format au format depot Git GitHub via envoie de votre URL à l'adresse suivante : lecocq@ipgp.fr avant la date et l'heure indiqué durant la scéance. N'oubliez pas de rediger un mail complet lors de votre envoie.

Utilisez le sujet suivant pour le mail : "[LP LPW 2022] compte rendu TP2 UNIX : Nom Prénom"

note : Tout fait partie de la note, le fond la forme, le respect des consignes, la citationn des sources ...

Pourquoi utiliser bash ?

Bash est une version évoluée du shell sh (le "Bourne shell"). Le shell peut être utilisé comme un simple interpréteur de commande, mais il est aussi possible de l'utiliser comme langage de programmation interprété (scripts).

La connaissance du shell est indispensable au travail de l'administrateur unix :
-le travail en "ligne de commande" est souvent beaucoup plus efficace qu'à travers une interface graphique
- dans de nombreux contextes (serveurs, systèmes embarqués, liaisons distantes lentes) on ne dispose pas d'interface graphique ;
- le shell permet l'automatisation aisée des tâches répétitives (scripts) ;
- de très nombreuses parties du système UNIX sont écrites en shell, il faut être capable de les lire pour comprendre et éventuellement modifier leur fonctionnement.

Autres versions de shell

Il existe plusieurs versions de shell : sh (ancêtre de bash), csh (C shell), ksh (Korn shell), zsh, etc. Nous avons choisi d'enseigner bash car il s'agit d'un logiciel libre, utilisé sur toutes les distributions récentes de Linux et de nombreuses autres variantes d'UNIX. Connaissant bash, l'apprentissage d'un autre shell sur le terrain ne devrait pas poser de diffi- cultés

Shell ou Python ?

Nous avons vu qu'il était possible d'écrire des programmes en shell. Pour de nombreuses tâches simples, c'est effectivement très commode. Néanmoins, le langage shell est forcément assez limité ; pour des programmes plus ambitieux il est recommandé d'utiliser des langages plus évolués comme Python ou Perl, voire des langages compilés (C, C++) si l'on désire op- timiser au maximum les performances (au prix de coûts de développement plus importants).

Les mauvais côtés des shell

Le shell possède quelques inconvénients :
- documentation difficile d'accès pour le débutant (la page de manuel "man bash" est très longue et technique) ;
- messages d'erreurs parfois difficiles à exploiter, ce qui rend la mise au point des scripts fastidieuse
- syntaxe cohérente, mais ardue (on privilégie la concision sur la clarté) ;
- relative lenteur (langage interprété sans pré-compilation). Ces mauvais côtés sont compensés par la facilité de mise en œuvre (pas besoin d'installer un autre langage sur votre système).

Scripts

Un script contient des séquences de commandes telles que l'on pourrait les taper dans un terminal. Les commandes successives peuvent être séparées par des retours à la ligne (qui sont interprétés comme des points virgules).
- Tout fichier de script commence par une ligne permettant d'identifier le programme qui doit être utilisé pour l'exécuter. Dans le cas d'un script bash, la première ligne du fichier doit contenir : `#!/bin/bash` (vérifiez le chemin de votre bash avec la commande which bash)
- Une ligne commençant par le caractère: `#` est considérée comme un commentaire et n'est pas exécutée par le shell. Comme toujours, il est très important de bien commenter son script pour qu'il soit compréhensible pour le reste du monde.
- Par convention, l'extension d'un script est : `.sh`
- Pour qu'un utilisateur puisse exécuter un script, il doit posséder les droits en exécution, mais aussi en lecture sur ce script (c'est un cas particulier où la lecture est nécessaire à l'exécution).
- Pour exécuter un script, on peut taper directement dans le terminal le chemin absolu du fichier, ou taper directement le chemin relatif du fichier en le faisant commencer par `./` ou taper la commande `bash` suivie d'un chemin du fichier.

Paramètres

Comme vous l'avez remarqué, la plupart des commandes Unix peuvent être suivies d'un ou plusieurs paramètres, qui peuvent être des options, des noms de fichiers ou de répertoires, etc. Il est aussi possible de passer des paramètres à vos scripts. Pour les manipuler, il existe plusieurs variables spéciales. En particulier :
- la variable `$#` contient le nombre de paramètres passés au script
- pour chaque entier i entre 1 et 9, la variable `$i` contient le i-ème paramètre
- la variable `$@` contient la liste de tous les paramètres séparés par des espaces
- la variable `$0` contient le nom du programme en cours d'exécution

Instructions

`If...then...elseif...else...fi` :

```
if liste1 ; then
commandes1
elseif liste ; then
...
...
else
```

```
...
...
fi
```

Le new line est équivalent au ;.
Si le code de retour de liste1 est 0,le bloc de commandes1 est exécuté, sinon on passe à la suite.
On peut aussi l'utiliser sous sa forme la plus simple :

```
if liste ; then
...
...
fi
```

```
while...do...done :
while liste ; do
...
...
done
```

Exemple :

```
while [ $# -ge 1 ] ; do
... # traitement de $1
shift
done
```

```
for...in...do...done :
for var in mot1 mot2 mot3 ;do ...
...
done
```

Exemple :

```
 Prog
 For var in 1 2 3 4 ;do
 echo $var
 done
```

```
% prog
1
2
3
4
%
```

```
ensemble=''element1 element2 element3''
...
...
for var in $ensemble ; do
...
...
done
```

Remarque :

```
for var in $* ;do
... # traitement de $var
done
est équivalent à
for var ;do
...
done
```

Les tests

Test de fichiers

Si le fichier existe et... :

-r :est lisible
-w :l'écriture est possible
-x :exécutable

Exemple :

```
if [ -r $2 ] ;then
...
...
else
echo "$0 :vous n'avez pas le droit de lire le fichier $2">&2
fi
```

-f :est un fichier ordinaire
-d :est un répertoire
-p :est une représentation interne d'un dispositif de communication
-c :est un pseudo-fichier du type accès caractère par caractère
-b :est un pseudo-fichier du type accès par bloc
-L :est un lien symbolique
-u : son Set UID=1
-g :son Set GID=1
-k :son Sticky Bit=1
-S :est non-vide

Tests de chaines

test chaîne (ou [chaîne]) : vraie si chaîne est une chaîne vide

-z chaîne :vraie si chaîne est une chaîne vide
-w chaîne :vraie si chaîne est une chaîne non-vide

Tests binaires

chaine1 = chaine2 : vraie si chaine1 est égale a chaine2
chaine1 != chaine2 : vraie si chaine1 n'est pas égale à chaine2

n1 -eq n2 : vraie si n1 est égal a n2
n1 -ne n2 : vraie si n1 est diffèrent de n2
n1 -gt n2 : vraie si n1 est plus grand strictement a n2
n1 -ge n2 : vraie si n1 est plus grand ou égal à n2
n1 -lt n2 : vraie si n1 est plus petit strictement a n2
n1 -le n2 : vraie si n1 est plus petit ou égal à n2

Fonctions

L'intérêt d'une fonction est que l'on peut en mettre plusieurs dans un script afin de gérer des répetitions de commandes (ou de groupes de commandes).

Déclaration :

```
nom(){
...
... ;
}
ou
nom(){.... ;}
```

A l'intérieur d'une fonction il est possible d'utiliser l'instruction return n : on quitte la fonction avec le code de retour n.

Appel :

```
% nom argument1 argument2 ... argumentn
```

shellcheck.net

finds bugs in your shell scripts.

Exercice : paramètres

Écrivez un script `analyse.sh` qui affiche :
Bonjour, vous avez rentré `nombre de paramètres` paramètres.
Le nom du script est `nom du script`
Le 3ème paramètre est `3ème paramètre`
Voici la liste des paramètres : `liste des paramètres`

Exercice : vérification du nombre de paramètres

Écrivez un script `concat.sh` qui prend en paramètre 2 mots et fait ce qui suit.
- si l'utilisateur rentre autre chose que 2 paramètres, indique à l'utilisateur qu'il doit rentrer exactement 2 paramètres, et quitte en renvoyant une erreur.
- sinon le script calcule dans une variable CONCAT la concaténation des 2 mots rentrés puis affiche le résulta

Pour tous les exercices suivants vous vérifirez systématiquement le nombre de paramètres.

Exercice : argument type et droits

Créer un script `test-fichier`, qui précisera le type du fichier passé en paramètre, ses permissions d'accès pour l'utilisateur, ou s'il n'existe pas.

Exemple de résultats :

```
Le fichier /etc est un répertoire
"/etc" est accessible par root en lecture écriture exécution

Le fichier /etc/smb.conf est un fichier ordinaire qui n'est pas vide
"/etc/smb.conf" est accessible par jean en lecture.
```

Exercice : Afficher le contenu d'un répertoire

Écrire un script bash `listedir.sh` permettant d'afficher le contenu d'un répertoire en séparant les fichiers et les (sous)répertoires.

Exemple d'utilisation :

```
$ ./listdir.sh /boot
```

affichera :

```
####### fichier dans /boot/
/boot/config-3.16.0-4-amd64
/boot/initrd.img-3.16.0-4-amd64
/boot/System.map-3.16.0-4-amd64
/boot/vmlinuz-3.16.0-4-amd64
####### repertoires dans /boot/
/boot/grub
```

Exercice : Lister les utilisateurs

Écrire un script bash affichant la liste des noms de login des utilisateurs définis dans `/etc/passwd` ayant un UID supérieur à 100.

Indication : `for user in $(cat /etc/passwd); do echo $user; done` permet preque de parcourir les lignes du dit fichier. Cependant, quel est le problème ? Résoudre ce problème en utilisant `cut` (avec les bons arguments) au lieu de `cat`. Faites la même chose avec la commande `awk`.

Exercice : Mon utilisateur existe t'il

Ecrire un script qui vérifie si un utilisateur existe deja.
- en fonction d'un login passé en paramètres
- en fonction d'un UID passé en paramètres
Si l'utilisateur existe renvoyer son UID à l'affichage.
Sinon ne rien renvoyer.

Exercice : Creation utilisateur

Ecrire un script pour créer un compte utilisateur voir : `man useradd`
Utilisez votre script de verification d'existance d'utilisateur avant de creer.
Il faudra vérifier que l'utilisateur en cours d'execution est bien root voir `echo $USER`
Il faudra créer son home dans /home aprés avoir vérifier qu'il n'y a pas déjà un repertoire portant le même nom.
Il faudra répondre à une suite de question : voir `man read`
- login
- Nom
- Prenom
- UID
- GID
- Commentaires

Exercice : lecture au clavier

La commande bash `read` permet de lire une chaîne au clavier et de l'affecter à une variable. Essayer les commandes suivante :

```
echo -n "Entrer votre nom: "
read nom
echo "Votre nom est $nom"
```

La commande `file` affiche des informations sur le contenu d'un fichier (elle applique des règles basées sur l'examen rapide du contenu du fichier).

Les fichiers de texte peuvent être affiché page par page avec la commande `more` (ou `less`, qui est légèrement plus sophistiquée, car *less is more...*).

- Question Tester les trois commandes : `read`, `file`, `more`.

- comment quitter `more` ?
- comment avancer d'une ligne ?
- comment avancer d'une page ?
- comment remonter d'une page ?
- comment chercher une chaîne de caractères ? Passer à l'occurence suivante ?

Écrire un script qui propose à l'utilisateur de visualiser page par page chaque fichier texte du répertoire spécifié en argument. Le script affichera pour chaque fichier texte (et seulement ceux là, utiliser la commande `file`) la question "voulez vous visualiser le fichier machintruc ?". En cas de réponse positive, il lancera `more`, avant de passer à l'examen du fichier suivant.

Exercice : appréciation

Créer un script qui demande à l'utilisateur de saisir une note et qui affiche un message en fonction de cette note :
- "très bien" si la note est entre 16 et 20 ;
- "bien" lorsqu'elle est entre 14 et 16 ;
- "assez bien" si la note est entre 12 et 14 ;
- "moyen" si la note est entre 10 et 12 ;
- "insuffisant" si la note est inférieur à 10.

Pour quitter le programme l'utiliateur devra appuyer sur q.