TP d'initiation au Shell

10 septembre 2019

Il existe plusieurs Shells, les plus connus étant *sh*, *bash*, *zsh*, ...Si les différences entre eux sont mineures, certaines questions de ce TP requièrent *bash*. Vous pouvez le lancer avec la commande `bash` dans un shell quelconque ; vous pourrez quitter à la fin du TP avec `exit`.

1 Manipulation de répertoires de fichiers

Lorsque vous ouvrez une session, votre répertoire de travail est votre dossier personnel. Vous avez vu en cours que la commande `pwd` (*Print Working Directory*) vous permet d'afficher le chemin absolu du répertoire actuel.

Q. 1: Vérifiez que votre répertoire de travail est bien votre `home` personnel (il est nommé selon votre identifiant).

Solution 1: Commande `pwd`.

Q. 2: La commande `ls` permet de lister le contenu d'un répertoire. Les options suivantes sont couramment utilisées :

`-a` : affiche les fichiers et dossiers cachés

`-l` : affiche les détails (type, droits, propriétaire, taille etc...)

Il est possible de grouper les options. Ainsi, la commande `ls -a -l` peut être écrite `ls -al`. Listez le contenu de votre `home`, avec puis sans les détails.

Solution 2: `ls ~ && ls -al ~`.

Q. 3: Affichez la liste des fichiers contenus dans le dépôt du groupe 1A (c'est un sous-dossier de `/home`), sans vous y déplacer. Proposez deux solutions, l'une avec un chemin absolu et l'autre avec un chemin relatif.

Solution 3:

Q. 4: Dans votre dossier `home` personnel, créez un répertoire nommé `TP_UNIX` à l'aide de la commande `mkdir` (*Make Directory*).

Solution 4: `cd ~ && mkdir TP_UNIX`

Q. 5: La commande `cd` (*Change Directory*) permet de se déplacer dans l'arborescence. Elle prend en paramètre un chemin, absolu ou relatif. Ce chemin peut être auto-complété à l'aide de la touche `TAB`. Entrez dans le répertoire créé à la question précédente.

Solution 5: `cd TP_UNIX`

2 Manipulation du contenu d'un fichier

Q. 6: Pour créer un fichier (vide), on peut utiliser la commande `touch`. Créez un fichier nommé *awesomefile.txt*.

Solution 6: `touch` awesomefile.txt

Q. 7: La commande `echo` permet d'écrire sur la sortie standard. Essayez-la avec le paramètre `coucou`.

Solution 7: `echo coucou`

Q. 8: On peut également rediriger la sortie de `echo` vers un fichier, ce qui permet d'écrire dans celui-ci. On utilise pour cela un chevron >. Cela écrase le contenu du fichier ; si l'on désire simplement ajouter du contenu à la fin du fichier, on utilise deux chevrons, >>. Écrivez *coucou* dans *awesomefile.txt*.

Solution 8: `echo coucou >` awesomefile.txt

Q. 9: Vérifier le résultat de la question précédente en utilisant la commande `cat`.

Solution 9: `cat` awesomefile.txt

Q. 10: `echo` n'est pas la seule commande dont la sortie peut être redirigée. Écrivez le manuel de la commande `cat`, dans *awesomefile.txt*.

Solution 10: `man cat >` awesomefile.txt

Q. 11: Pour des fichiers dépassant une certaine taille, la commande `cat` est peu pratique. Vérifiez le résultat de la question précédente à l'aide de `less`.

Solution 11: `less` awesomefile.txt

Il existe également des éditeurs de textes accessibles en console ; on peut citer *nano*, *vim*, ou *emacs*.

3 Manipulation de fichiers

Q. 12: À l'aide de la commande `cp` (*copy*), copiez *awesomefile.txt* dans votre `home`. Vérifiez le résultat avec `ls`.

Solution 12: `cp awesomefile.txt ../copied.txt && ls ..`

Q. 13: La commande `mv` (*move*) permet de déplacer/renommer des fichiers. Renommez le fichier que vous venez de copier dans votre `home`, sans oublier de consulter le manuel.

Solution 13: `mv ../awesomefile.txt ../newName.txt`

Q. 14: Supprimez maintenant ce fichier, à l'aide de `rm` (*remove*).

Solution 14: `rm ../newName.txt`

Q. 15: Qu'aurait-il fallu changer dans les questions précédentes si l'on avait manipulé un dossier (vide/non vide) ?

Solution 15:

`cp` : rajouter `-r`

`mv` : rien

`rm` : rajouter `-r`, ou utiliser `rmdir` si vide

4 Gestion des droits d'accès

Pour gérer l'aspect multi-utilisateurs, les fichiers/dossiers Unix disposent de droits d'accès :

r lecture

w écriture

x exécution

Ceux-ci peuvent être réglés différemment suivant trois niveaux :

u l'utilisateur

g le groupe, sauf l'utilisateur

o tout le monde, sauf le groupe et l'utilisateur

Pour un dossier, le droit d'exécution correspond au droit d'accès, tandis que les droits de lecture et d'écriture correspondent à l'accès à la liste des fichiers s'y trouvant et à la création de nouveaux fichiers.

Pour visualiser ces droits, on utilise la commande `ls -l`. On obtient alors un résultat de la forme de la figure 1. On s'intéresse alors à la première colonne, à lire suivant les informations de la figure 2.

Pour changer les droits d'un fichier/dossier, on utilise la commande `chmod`.

Q. 16: Créez un fichier nommé *test.sh*, et écrivez-y la commande `echo coucou` (par la méthode de votre choix). *test.sh* est alors un script shell, potentiellement exécutable, qui écrira "coucou" à chaque fois qu'il sera lancé (par la commande `./test.sh`). Notez les droits actuels du fichier. Avez-vous ceux nécessaires pour l'exécuter ?

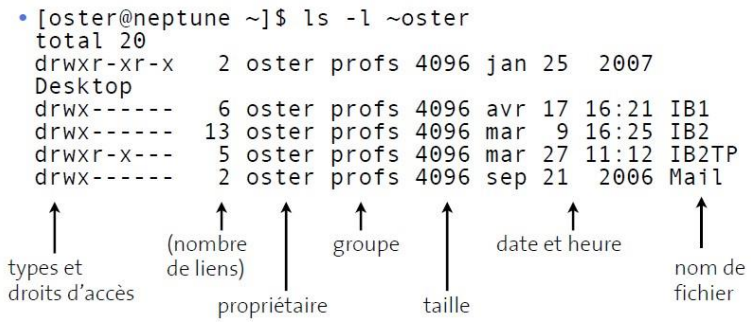

FIGURE 1 – Résultat de `ls -l`

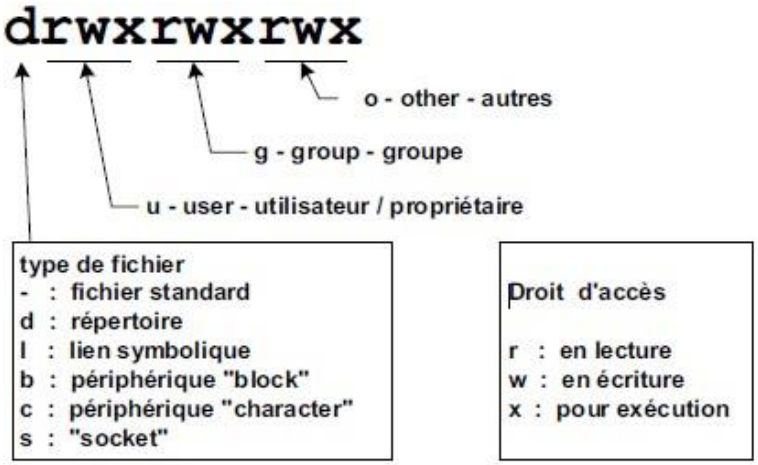

FIGURE 2 – Légende des droits d'accès

Solution 16: `echo "echo coucou" > test.sh`
`ls -l`
`-rw-r--r--`
Non.

Q. 17: Il existe plusieurs manières d'utiliser `chmod`. Entrez la commande suivante :
`chmod u+x test.sh`. Que remarquez-vous ? Consultez le manuel et expliquez `u+x`.

Solution 17: L'utilisateur peut maintenant exécuter `test.sh`.

Q. 18: Vous avez sans doute remarqué dans le manuel la deuxième manière d'utiliser `chmod`. Il s'agit de l'écriture *octale* des droits d'accès. Entrez la commande `chmod 777 test.sh`, puis visualisez les nouveaux droits de `test.sh`. Que remarquez-vous ?

Solution 18: Tout le monde a tous les droits.

5 Gestion des processus

Q. 19: Que fait la commande `ps` ? Essayez-la, ainsi qu'avec l'option `-e`.

Solution 19: `ps` liste les processus en cours.

Lorsque l'on lance une commande, le shell ne redonne pas la main avant que celle-ci ne se soit terminée. On peut cependant terminer ou arrêter momentanément l'exécution d'une commande, à l'aide des raccourcis suivants : **Ctrl-c** (fin) et **Ctrl-z** (pause). Le shell rend alors la main.

Si le processus a été mis en pause, il est alors possible de relancer son exécution, en premier plan (on reprend alors la main) ou en arrière plan (on garde alors la main). On utilise pour cela les commandes `bg` (*background*) et `fg` (*foreground*).

Q. 20: Lancez la commande `cat` (sans aucun argument). Que remarquez-vous ? Mettez le processus en pause, puis relancez-le avant de le terminer, à l'aide des commandes/raccourcis vus plus haut.

Solution 20: `cat`, puis **Ctrl-z**, puis `fg`, puis **Ctrl-c**.

Il est également possible de lancer une commande directement en arrière-plan, en la terminant avec le signe `&`.

On peut utiliser la commande `jobs` pour savoir quels processus sont actuellement en arrière plan.

6 Bonus : Les | (pipes)

Le symbole | sert à chaîner les entrées/sorties de différents processus. Ainsi, en écrivant `proc1 | proc2`, la sortie de `proc1` est chaînée à l'entrée de `proc2`.

Q. 21: Il est courant d'utiliser les *pipes* en conjonction avec la commande `grep`. A quoi sert-elle ? Obtenez la liste des processus dont le nom contient la lettre `x`.

Solution 21: `ps -e | grep x`.

7 Bonus : Retrouver des commandes précédemment écrites

Q. 22: On peut vite être limité par l'utilisation des touches flèches haut et bas pour retrouver d'anciennes commandes. La commande `history` affiche tout l'historique des commandes que vous avez rentrées. Utilisez la conjointement avec `grep`, pour trouver toutes les commandes que vous avez rentrées utilisant `ls`.

Solution 22: `history | grep ls`.

Q. 23: `!!` correspond à la dernière commande entrée. Utilisez-le avec `echo` pour afficher cette dernière (sans l'exécuter).
NB. : cela est utile lors d'un oubli de `sudo`, qui permet de lancer une commande en tant qu'administrateur, avec `sudo !!`.

Solution 23: `echo !!`.

Q. 24: La plupart des shell (dont *bash*) proposent aussi le raccourci **Ctrl-r**. Utilisez-le pour retrouver votre dernière commande `cat`. Lors de votre recherche, si vous voulez accéder à des résultats plus anciens, vous pouvez retaper **Ctrl-r**.

Solution 24: **Ctrl-r**, puis `cat`

8 Bonus : Les variables d'environnement

Q. 25: Il est possible de définir des variables en dehors des scripts, à l'aide des commandes `export` ou `setenv`. Définissez une variable nommée `hello`, et donnez-lui la valeur `"Hello World!"`. Comment afficher cette variable ? Quelle est la durée de vie et la portée des variables définies ainsi ?

Solution 25: `export hello="Hello World!"` ou `setenv hello "Hello World!"`. Affichage avec `echo $hello`.
Durée de vie : la session du Shell
Portée : le terminal courant

Q. 26: Certaines variables sont prédéfinies. C'est le cas de `OLDPWD`. Que vaut-elle ? Déplacez-vous. Que remarquez-vous ?
NB. : cette variable est ensuite surtout utilisée lorsque l'on tape `cd -`.

Solution 26: Répertoire de travail précédent.

9 Bonus : Divers

Q. 27: Le symbole ~ est synonyme de `home`. Utilisez-le pour lister le contenu de votre `home`, à partir de la racine (`/`).
NB. : `cd ~` est donc équivalent à `cd` (sans arguments).

Solution 27: `ls ~`

Q. 28: Essayez la commande `clear` et constatez son effet, puis essayez avec son raccourci **Ctrl-l** (fonctionne avec bash)

Solution 28: Effet observé : le terminal est vidé de toutes commandes qui sont "poussées vers le haut" et le prompt est en haut de la fenêtre du terminal

Q. 29: À l'aide de la commande `which`, retrouvez le chemin de la commande `file`. À quoi sert cette dernière ?

Solution 29: `which file`
`file` permet de connaître le type d'un fichier, (un peu) plus sûrement qu'à partir de son extension.

Q. 30: Utilisez la commande `find` pour retrouver tous les fichiers `jar` contenus dans le dépôt 1A. *(Aide : utilisez `man 7 glob`. Attention cette entrée du manuel n'existe pas sous MacOS)*

Solution 30: `find /home/depot/1A -name *.jar`

10 Pour les plus téméraires :

Q. 31: Rendez-vous sur : `http://overthewire.org/wargames/bandit/`