Le shell pour tous

- Objet : Définition et liste des Shell.
- Niveau requis :
  débutant, avisé
- Commentaires : *Qu'est-ce qu'un shell, découverte et prise en main d'icelui.*
- Débutant, à savoir : Utiliser GNU/Linux en ligne de commande, tout commence là !.
- Suivi :
  à-completer, à-tester
  - Création par captnfab 30/09/2013
  - Testé par <...> le <...>
- Commentaires sur le forum : ici [1)]

Nota :

Contributeurs, les Fix Me! sont là pour vous aider, supprimez-les une fois le problème corrigé ou le champ rempli !

Introduction

Le *shell*, également appelé *interpréteur de commandes*, est l'outil le plus **élémentaire**, direct et immédiat pour manipuler les fichiers et lancer des programmes.

Il permet par exemple :

- de lister les fichiers d'un dossier grâce au programme ls;
- de créer un répertoire avec le programme mkdir
- etc...

En pratique, le shell permet de lancer et de faire interagir entre eux tous les programmes installés sur la machine, le plus souvent regroupés dans les dossiers `/usr/bin`, `/bin`, `/usr/sbin` et `/sbin`.

Le premier *interpréteur de commandes* était **sh** dans l'UNIX original de 1971. Afin d'améliorer la productivité, plusieurs revendeurs (HP, Sun...) dans les années 1970 et 1980 ont créé (ou fait créer) d'autres interpréteurs (csh, ksh...) dont les règles de grammaire plus poussées facilitaient l'écriture de commandes complexes ou répétitives, et accéléraient l'exécution de *scripts* en prenant en charge certaines actions élémentaires (echo, test, [, ...) autrefois déléguées à des programmes de **/bin**.

La majorité de ces améliorations successives ont été reprises dans **bash** qui est le shell utilisateur présent par défaut sur Debian. **zsh** est une alternative à bash également très utilisée. Enfin, **dash** est le shell léger utilisé par défaut par Debian pour exécuter les scripts shell.

Lancer un shell

Pour lancer un shell, il suffit d'ouvrir un émulateur de terminal, comme Gnome-Terminal, XTerm, Rxvt-Unicode, etc. Ce dernier exécutera automatiquement votre shell utilisateur.
Il est aussi possible d'ouvrir une des consoles virtuelles du système, par une combinaison de touches

telle que Ctrl+Alt+F1 ou en démarrant le système en mode non graphique.

Obtenir de l'aide

Une des commandes les plus importantes à lancer dans un shell est la commande `man` qui permet d'afficher l'aide d'un programme. Par exemple, jetez un coup d'œil à

```
man bash
```

(touche Q pour quitter)

Le prompt

Le *prompt* ou invite de commande est le petit texte qui est déjà affiché quand on lance le shell et qui reste affiché quand on appuie sur la touche ↵ Entrée

Souvent, le prompt ressemble à ça : `nom-d-utilisateur@nom-de-la-machine:dossier-courant$`.

Le dossier courant pouvant être abrégé en ~ s'il s'agit du dossier personnel (par défaut).

Lancer un programme

Programmes, paramètres et arguments

Pour lancer un programme, rien de plus simple, il suffit de taper son nom juste à droite du prompt, et de valider par entrée. Exemple :

```
pwd
```

Après cela, pwd affiche le dossier courant, probablement

```
/home/nom-d-utilisateur
```

puis il termine. Et rend la main, c'est-à-dire que le prompt est à nouveau visible.

Il est possible de donner un **argument** à un programme. Par exemple, pour lister le contenu (`ls`) du dossier `/var`, on tape :

```
ls /var
```

Là encore, le programme `ls` affiche le contenu du dossier et puis rend la main au shell.

Essayez donc :

```
ls -l /usr/bin
```

Vous voyez que `-l` a modifié l'affichage de `ls`. On dit que l'argument `-l` est un **paramètre**, les paramètres d'un programme étant listés et expliqués dans le `man` de ce programme. Vous pouvez donc vous précipiter sur :

```
man ls
```

Un programme est accessible par appel via son nom s'il est installé dans un répertoire indiqué dans le path. Nous vous conseillons de lire le paragraphe sur les variables d'environnement avant de tenter de le modifier.

Lorsque les arguments à donner aux programmes sont des fichiers, il peut être intéressant d'utiliser les méta-caractères. Fix Me![2)]

- Une liste de commandes disponibles est présenté ici gnu_linux, une autre ici le_debianiste_qui_papillonne.

Naviguer dans les fichiers et les dossiers

Pour une découverte pas à pas mais en profondeur, voir Chemin relatif ou absolu illustrés par quelques commandes

- Les commandes de bases de la navigation dans le système de fichier pour copier, déplacer, renommer, lister, supprimer ou chercher les fichiers et les dossiers.
- S'y retrouver dans l'arborescence avec les chemins relatifs et absolus
- Comprendre l'arborescence des fichiers d'une distribution Debian fhs-accueil
- Transférer des fichiers à distance
  - sur une autre machine sous Linux avec scp
  - sur un partage de fichiers Windows avec smbget

Bien sûr, la page `man` de chacun de ces programmes (sauf pour `cd` qui est une commande du shell et non un programme) est une source précieuse d'information !

Voici un petit exemple pour les gens pressés :

- Je me place dans /usr/

```
cd /usr
```

- Je me place dans /usr/bin

```
cd bin
```

- Je liste les fichiers et dossiers de /usr/bin

```
ls
```

- Je me place dans /tmp

```
cd /tmp
```

- Je crée un dossier /tmp/coucou

```
mkdir coucou
```

- Je crée un dossier /tmp/coucou/hop

```
mkdir coucou/hop
```

- Je crée un fichier /tmp/coucou/pouet

```
touch coucou/pouet
```

- Je renomme “pouet” en “bla”

```
mv coucou/pouet coucou/bla
```

- Je supprime le dossier vide /tmp/coucou/hop

```
rmdir coucou/hop
```

- Je supprime le dossier /tmp/coucou et son contenu

```
rm -r /tmp/coucou
```

- Je retourne dans mon dossier personnel

```
cd
```

Le tilde (~)

Le caractère ~ permet d'indiquer le nom du répertoire d'accueil d'un utilisateur :

```
~
```

“répertoire d'accueil du propriétaire du shell”.

et :

```
~username
```

“répertoire d'accueil du compte username”.

Rediriger l'affichage

Dans un shell, il faut penser *flux*. Un flux est une sorte de fichier, qui se construit au fur et à mesure et n'est présent qu'en mémoire. Par exemple,

- tout ce qu'affiche un programme (comme `ls` ou pwd est un flux, affiché directement sur le terminal).
- tout ce que tape un utilisateur pendant qu'il utilise un programme du terminal est un flux.

Cela vaut en particulier pour le shell ! Vous pouvez le vérifier en tapant la combinaison de touches correspondant à une *fin de fichier* (ou fin de flux) : Ctrl+D **paf** le shell se ferme, parce que son flux d'entrée est terminé

Chevrons et redirections de flux

Il est possible de rediriger un flux vers un fichier. Par exemple, pour enregistrer un fichier `/tmp/liste.txt` contenant la liste des fichiers et dossiers du répertoire `/usr/bin`, c'est aussi simple que

```
ls /usr/bin > /tmp/liste.txt
```

Pour en savoir plus, voir les chevrons

Le tuyau (pipe) |

Admettons que vous vouliez lancer un premier programme, qui fournit alors un flux de sortie, puis donner ce flux de sortie à un deuxième programme pour qu'il y fasse un second traitement.

Par exemple, vous voulez lister l'ensemble des fichiers dans `/usr/bin` (via `ls`) mais vous ne voulez garder seulement que les fichiers comportant `term` dans leur nom. Vous utiliserez alors la commande `grep` pour faire la 2e opération, qui fonctionne à la manière d'un filtre :

```
ls /usr/bin | grep 'term'
```

Magique, non ? :)

Pour comprendre son fonctionnement interne :

- voir le tuto sur le pipe.

Arguments dynamiques

Parfois, vous pouvez vouloir utiliser la sortie d'un programme comme argument d'un autre programme. Pour ce faire, on utilise `$( commande )`. Exemple, vous voulez écrire une commande qui affiche (la commande `echo` sert à cela) :

> Coucou, je suis dans le répertoire *nom-du-répertoire* et je m'amuse comme un fou.

Vous savez alors qu'il vous suffit de taper le code suivant :

```
echo 'Coucou, je suis dans le répertoire' "$(pwd)" "et je m'amuse comme un fou."
```

Notez qu'ici, nous avons donné trois arguments à `echo` :

- Le premier argument est du texte simple, ils est donc entre guillemets simples : ' .
- Le second argument comporte un code qui doit être interprété par le shell `$(pwd)` il doit donc donner entre guillemets doubles : “ .
- Pour ne pas confondre l'apostrophe du troisième argument avec la fin d'un argument, il est indiqué entre guillemets doubles également.

Alias

Attention, les alias ne font pas partie du standard POSIX, autrement dit, ils peuvent fonctionner de manière différente d'un shell à l'autre. En particulier, vous ne **devez pas les utiliser dans des scripts.** Voir http://rgeissert.blogspot.com/2013/09/a-bashism-week-aliases.html

Lorsque vous tapez une ligne de commande compliquée assez régulièrement, il est intéressant de la rédiger une fois pour toute dans un fichier et de la rappeler ensuite rapidement via un mot-clé. C'est ce que l'on appelle un alias. Ils sont gérés de manière légèrement différentes par chacun des shells. La syntaxe générale est la suivante :

```
alias irc='x-terminal-emulator -title irc -name irc -e weechat'
```

```
alias cdmonproj='cd ~/projets/debian-facile/2013/mon-projet/src/www/'
```

Pour que les alias soient actifs dans tous les terminaux, il suffit de les placer dans le `~/.bashrc` (si vous êtes sous `bash`, ou `~/.zshrc` pour `zsh`, etc.

Pour plus de détails suivant les shells :

- utiliser les alias avec bash.

Variables et environnement

Dans un shell, il y a tout un tas de variables qui lui sont transmises par le processus l'ayant lancé. On les appelle les variables d'environnement, et elles sont listées par la commande `env` :

```
env
```

Pour définir une variable, c'est facile :

```
NOM="Capitaine Fab"
```

Et pour l'afficher :

```
echo "$NOM"
```

Pour que la variable soit transmise à l'environnement des programmes qui seront lancés par le shell, il faut l'exporter :

```
export NOM
```

Les opérations de définition et d'export de la variable peuvent être effectuées simultanément :

```
export NOM="Capitaine Fab"
```

Les possibilités sont énormes, voir le tuto dédié :

- variables Les variables en détail.

Rédaction de scripts shell

Il est possible d'automatiser les tâches avec des scripts shell. Vous apprendrez notamment à utiliser les boucles et les instructions conditionnelles, et les fonctions.

- Rédaction de scripts shell
- Fonctionnalités avancées des scripts shell

Garder la main

Parfois, vous lancez un programme dans le shell, mais vous voulez continuer à utiliser ce dernier pendant que le programme tourne en *arrière-plan*.

Pour ce faire, on utilise l'esperluette :

```
iceweasel &
```

Avec cela, iceweasel va être lancé, mais vous aurez toujours la main dans le shell (tape Entrée pour le vérifier.)

Oups, j'ai oublié de mettre le &, comment je fais ?

1. D'abord, il faut envoyer un SIGSTOP au programme en appuyant sur Ctrl+Z
2. Ensuite, il faut indiquer au programme de reprendre sa marche, mais en arrière plan, avec la commande bg.

Ne pas fermer un programme en même temps que le shell

Pour lancer un programme en arrière-plan de sorte à ce qu'il ne se ferme pas en même temps que le shell, il faut le lancer via

```
iceweasel &!
```

Oups, j'ai oublié de mettre le `!`, comment je fais ?

Il suffit de taper `disown leprogramme` (faire Tab ⇆ pour voir les propositions)

Enchaîner plusieurs commandes

Plusieurs commandes sur la même ligne

On peut exécuter plusieurs commandes sur la même ligne en les séparant par un **;**

On peut reprendre l'exemple Naviguer dans les fichiers et les dossiers sur une seule ligne

```
cd /usr ; cd bin ; ls ; cd /tmp ; mkdir coucou ; mkdir coucou/hop ; touch coucou/pouet ;
```

```
mv coucou/pouet coucou/bla ; rmdir coucou/hop ; rm -r /tmp/coucou ; cd
```

Contrairement à ce qui va suivre, la commande suivante va s'enchaîner quelque soit la façon dont s'est déroulée la commande précédente.
Imaginons par exemple que le dossier *tmp* n'existe pas. La commande *cd /tmp* ne pourra donc pas être exécutée. Cependant, toutes les autres commandes vont l'être. Dans le cas présent, elles échoueraient toutes, puisque vous essayerez d'écrire dans le dossier */usr/bin* qui appartient à *root*.

En fonction de leurs codes retour avec && et ||

Il peut être parfois intéressant d'exécuter une commande en fonction de la bonne exécution ou non de la commande précédente. Cela est possible grâce au **code retour** des commandes.

Qu'est ce que le code retour

Le code retour d'une commande permet de savoir si celle-ci a été exécutée sans problème. Dans ce cas, le code retour est **0**. Sinon, elle peut prendre n'importe quel autre valeur.

Il ne faut pas confondre le code retour de la commande avec son résultat. Le résultat est ce qui s'affiche à l'écran. Le code de retour est une valeur numérique qui peut être utilisé dans un test par exemple.
On peut afficher le code retour de la dernière commande exécutée avec

```
echo $?
```

Les opérateurs && et ||

Cette ligne de commandes permet d'exécuter *cmd2* si *cmd1* s'est exécutée correctement (si son code retour est 0) :

```
cmd1 && cmd2
```

Cette ligne de commandes permet d'exécuter *cmd2* si *cmd1* ne s'est pas exécutée correctement (si son code retour est différent de 0) :

```
cmd1 || cmd2
```

Par exemple mettre à jour le système après avoir mis à jour la liste des paquets sans problème tapez :

```
apt-get update && apt-get upgrade
```

Autre exemple, pour créer le fichier *coucou* dans */usr/tmp* si celui-ci n'existe pas tapez :

```
ls /usr/tmp/coucou || touch /usr/tmp/coucou
```

On peut combiner ces opérateurs. Il suffit de se rappeler qu'il vont prendre en compte le code retour de la dernière commande exécutée.

Ces opérateurs fonctionnent un peu comme des tests *if*. Pour plus de précision sur les tests, on peut consulter : Fonctionnalités avancées du Shell

Quelques raccourcis en shell Linux

Quelques raccourcis en shell Linux (à apprendre par cœur 😉) :

Raccourci clavier	Action
Ctrl+A	Déplace le curseur au début de la ligne
Ctrl+E	Déplace le curseur à la fin de la ligne
Ctrl+K	Efface du curseur à la fin de la ligne
Ctrl+U	Efface la ligne jusqu'au curseur. Efface donc la ligne si le curseur se trouve à la fin
Ctrl+L	Efface le terminal, équivalent) `clear`
Ctrl+W	Effacer du caractère précédent le curseur jusqu'au début du mot
Alt+←	Comme Ctrl+W
Alt+D	Coupe la chaîne depuis le caractère situé sous le curseur jusqu'à la findu mot (si le curseur est placé au début d'un mot, coupe le mot)
Ctrl+Y	Colle la sélection précédemment coupée
Ctrl+T	Inverse les deux caractères précédent le curseur
Alt+T	Inverse deux mots précédent le curseur
Alt+C	Met en majuscule la lettre située sous le curseur et déplace le curseur à la fin du mot
Alt+L	Met en minuscule toutes les lettres depuis la position du curseur jusqu'à la fin du mot
Alt+U	Met en majuscule toutes les lettres depuis la position du curseur jusqu'à la fin du mot
Alt+_	Annule la modification précédente

A noter que ces raccourcis claviers sont identiques à ceux utilisés dans l'éditeur de texte *emacs*. Il est possible d'obtenir un comportement identique à celui de *vi/vim* avec

```
set -o vi
```

La ligne de commande est alors en mode insertion. Pour passer en mode normal permettant de se déplacer dans une ligne saisie avec les raccourcis habituels de vim, c'est Esc. Pour entrer en mode insertion, c'est A ou I.

Pour retrouver les raccourcis par défaut

```
set -o emacs
```

si vous utilisez zsh

Raccourci clavier	Action
Echap+H	Aide de la commande en cours de saisie
Echap+Q	Efface la ligne courante, mais la ré-affichera une fois la commande courante terminée(Ma préférée 😃)

Tableau des Shell disponibles

SHELL	COMMENTAIRES
bash	Le shell utilisateur par défaut sous Débian
dash	Le shell système par défaut sous Débian
zsh	Un shell utilisateur très paramétrable (Le SHELL ULTIME :))

1)

N'hésitez pas à y faire part de vos remarques, succès, améliorations ou échecs !

2)

Il faut supprimer la partie méta-caractères du tuto regexp et le déplacer ailleurs.