Sommaire

Le shell ou le retour du C:> ! (et les commandes)

Le shell ou le retour du C:> !
(et les commandes)
par Jean−Christophe, Marc et Anne
Le shell, un environnement écrit au temps ou les hommes étaient des hommes :−)

Introduction

Qu'est−ce que **shell** me direz vous ? Certains diront que c'est ça le vrai Linux. il n'y a pas que du faux là−dedans, puisque étymologiquement parlant, "Linux" est juste le nom du noyau du système d'exploitation, et qu'on a tendance par abus de langage à utiliser "Linux" pour désigner l'ensemble de Linux, du serveur X et des nombreuses applications.

Bref, le shell c'est le bon vieux *mode texte*, mon copain le *prompt*, qui sous Linux revêt une importance capitale. En effet, la philosophie Unix veut que toute action puisse être réalisée en ligne de commande, avant d'être accessible dans une boîte de dialogue. Ainsi de nombreuses applications X ne sont en fait que des *front ends* (des façades) à des applications en ligne de commande, se contentant de construire la bonne ligne de commande à partir de vos clics (XCDRoast / cdrecord, mkisofs, etc. ; kppp / pppd ; etc.).

Comme Linux prône le règne de la liberté, vous n'avez pas qu'un seul *shell* disponible. Vous pouvez utiliser **bash**, **tcsh**, **ksh**, **ash**, **sh**, **csh**, etc. Néanmoins, la plupart des distributions actuelles proposent **bash** par défaut, et je vous recommande donc de l'utiliser, surtout si vous débutez sous Unix et que vous n'avez pas encore d'habitudes. Si plus tard, vous tombez sur un ordinateur ne disposant que de tcsh, ne vous inquiétez pas : la différence n'est pas flagrante, et vous pourrez toujours consulter "man tcsh" !
à noter que le choix du shell pour un utilisateur se configure dans /etc/passwd.

Le shell n'est pas seulement le prompt vous permettant de taper vos commandes, c'est aussi un puissant *langage de commande*, vous permettant d'automatiser des tâches, etc. via l'écriture de *scripts shell*. Apprendre le langage du shell peut être très enrichissant et utile ; néanmoins, cela dépasse le cadre de cette rubrique. Et pis j'ai pas le courage de taper une leçon sur bash :) Par contre, vous trouverez ici les commandes de base, ainsi que les raccourcis clavier et les raccourcis du shell, les aliases, les variables d'environnement, la configuration du shell et j'en passe. C'est déjà pas mal non ?

Commandes pour débuter

Avant de commencer, il faut savoir que Linux est **sensible à la casse** (*case sensitive* en anglais), c'est à dire qu'il distingue les majuscules des minuscules. Ainsi, si je vous dit que la commande est 'mkdir', ce n'est pas la peine d'essayer MKDIR ou mKdiR, cela ne fonctionnera pas. De même, les noms de fichiers et de répertoires sont également sensibles à la casse.

De plus, sous Unix, les chemins sont séparés par des slash : écrivez /etc/rc.d/init.d/xfs mais jamais etc\rc.d\init.d\xfs par pitié :)

- **Répertoires spéciaux** :

. représente le répertoire courant, .. représente le répertoire parent ~ représente le répertoire maison (home) de l'utilisateur

- **Fichiers cachés** :

sous Unix, les fichiers cachés commencent par un point. Par exemple, ~/.bashrc est un fichier caché, dans le répertoire maison de l'utilisateur, qui contient la configuration de son shell.

- **Jokers** : ? et *

Les caractères ? et * dans les noms de fichiers et de répertoires permettent de représenter des caractères quelconques. '?' représente un seul caractère, tandis que '*' en représente un nombre quelconque. Par exemple "*.jpg" représente tous les fichiers se terminant par jpg ; "*toto*" tous les fichiers contenant "toto". Oui vous avez bien vu : on peut mettre plusieurs étoiles en même temps !!! Vous pouvez même faire : cd /et*/rc.*/init*, cela risque de fonctionner !!! Il faut également savoir que c'est le shell qui interprète ces caractères avant de transmettre la ligne de commande. Par exemple, si vous tapez : rm −Rf *.tmp, le shell transformera cette ligne de commande en : rm truc1.tmp truc2.tmp truc3.tmp.

- **Jokers avancés** : []

Vous pouvez aussi utiliser les crochets pour spécifier des caractères : [a] signifie : égal à 'a'. Exemple : rm *[a]* efface tous les fichiers contenant la lettre 'a'. [!a] signifie : différent de 'a'. Exemple : rm *[!a]* efface tous les fichiers, sauf ceux contenant la lettre 'a'. [abc] signifie : l'un des caractères a, b ou c. Exemple : rm [abc]*.tmp efface tous les fichiers commençant par a, b ou c. [a−l] : signifie : tous les caractères compris entre a et l. Exemple : rm fic_[a−l]* efface tous les fichiers commençant par fic_ suivi d'une quelconque lettre entre a et l.

Bon c'est pas tout ça, voici les commandes de base sous Linux :

Commandes linux	équivalent MsDos	à quoi ça sert	Exemples :
cd	**cd**	change le répertoire courant.	cd .. − va dans le répertoire parent du répertoire courant cd /home/user/.nsmail − va dans le répertoire désigné
ls	**dir**	affiche le contenu d'un répertoire	ls − affiche le contenu du répertoire courant ls −l − affiche le contenu du répertoire courant de manière détaillée ls −a /home/user − affiche le contenu du répertoire désigné (ainsi que les fichiers cachés)
cp	**copy** **xcopy**	copie un ou plusieurs fichiers	cp toto /tmp − copie le fichier toto dans le répertoire /tmp

			cp toto titi − copie le fichier toto sur le fichier titi cp −R /home/user /tmp/bak − copie le répertoire /home/user ainsi que tout ce qu'il contient dans /tmp/bak
rm	**del**	efface un ou plusieurs fichiers	rm toto titi − efface les fichiers toto et titi rm −f toto titi − efface les fichiers toto et titi sans demander confirmation
rm −rf	**deltree**	efface un répertoire et son contenu	rm −rf /tmp/* − efface (sans demander de confirmation) tous les fichiers et répertoire de /tmp
mkdir	**md**	crée un répertoire	mkdir /home/user/mes documents − crée le répertoire "mes documents" dans le sous répertoire /home/user
rmdir	**rm**	efface un répertoire s'il est vide	rmdir /home/user/.nsmail − efface le répertoire .nsmail de /home/user si celui−ci est vide
mv	**ren** **move**	déplace ou renomme un ou des fichiers	mv tata titi − renomme tata en titi mv * *.bak − ne fonctionne pas !!!! mv * /tmp/bak − déplace tous les fichiers du répertoire courant vers le répertoire /tmp/bak
find	**dir −s**	trouve un fichier répondant à certains critères	find /home −name "*bash*" − trouve tous les fichiers contenant le mot bash dans leur nom se trouvant dans le répertoire /home
locate	**dir −s**	trouve un fichier d'après son nom	locate bash − trouve tous les fichiers contenant le mot bash dans leur nom complet (avec le répertoire) : à la différence de find, locate trouve ses informations dans une base de donnée créée par updatedb
man	**help**	affiche l'aide concernant une commande particulière	man ls − affiche l'aide (page de manuel) de la commnade ls. On quitte man en appuyant sur la touche 'q'
chmod	pas	modifie les permissions d'un	chmod o+r /home/user

	d'équivalent	fichier	− autorise les autres (o=other) (ie: ceux qui ne sont ni le propriétaire, ni membre du groupe propriétaire) à lire (r=read) le répertoire /home/user chmod a+rw /home/user/unfichier − autorise tout le monde (a=all) à lire et écrire (w=write) dans le fichier /home/user/unfichier
chown	pas d'équivalent	modifie le propriétaire d'un fichier	chown user unfichier rend user propriétaire de unfichier.
chgrp	pas d'équivalent	modifie le groupe proprétaire d'un fichier	chgrp −R nobody /home/httpd − rend le groupe : nobody (un groupe ayant très peu de droit sur un système linux) propriétaire de /home/httpd ainsi que tout les fichiers qu'il contient (−R)
ln −s	pas d'équivalent	crée un lien vers un fichier	ln −s /dev/fd0 /dev/disquette crée un lien vers /dev/fd0 (le lecteur de disquette) nommé /dev/disquette. La manipulation de /dev/fd0 et /dev/disquette (sauf l'effacement).
grep	pas d'équivalent	recherche une chaine dans un fichier (en fait recherche une expression régulière dans plusieurs fichiers)	grep chaine *.txt − recherche la chaine 'chaine' dans tous les fichier se terminant par .txt.
which	pas d'équivalent	trouve le répertoire dans lequel se trouve une commande	which emacs − retourne le nom du répertoire dans lequel se trouve la commande emacs.
cat	**type**	affiche un fichier à l'écran	cat ~/.bashrc − affiche le contenu du fichier ~/.bashrc

Remarque :
Pour en savoir plus sur toutes ces commandes, je vous conseille de consulter leur page de man !

La commande ls

Cette commande est omniprésente, aussi il est bon d'en présenter les basiques.

Afficher le listing page par page : ls | less (less est une version améliorée de more)
Afficher le listing en couleurs : ls −−color
Afficher aussi les fichiers cachés (commençant par un point) : ls −a
Mettre un '/' après les noms de répertoires : ls −p
Afficher le listing détaillé : ls −l

Tri sur la date
Pour afficher les fichiers d'un répertoire en triant sur la date de mise à jour des fichiers
Afficher les fichiers les plus récents en premier : ls −t

Afficher les fichiers les plus vieux en premier : ls −rt
Mixer avec l'option "l" afin d'afficher le listing détaillé : ls −rtl ou ls −tl

bien sûr, toutes ces options sont mixables, ainsi "ls −altp" affiche tous les fichiers, de façon détaillée, dans l'ordre chronologique, an ajoutant '/' après chaque nom de répertoire.

Exemple de listing

[jice@taz jice]$ls −alp
total 144
−rw−r−−r−− 1 jice users 24 Aug 2 21:37 .bash_logout
−rw−r−−r−− 1 jice users 230 Aug 2 21:37 .bash_profile
−rw−r−−r−− 1 jice users 467 Aug 2 21:37 .bashrc
−rw−r−−r−− 1 jice users 1452 Aug 2 21:37 .kderc
drwxr−−r−− 12 jice users 1024 Aug 2 21:37 .kde/
drwxr−−r−− 2 jice users 1024 Aug 2 21:37 Desktop/
−rw−r−−−−− 1 jice users 1728 Aug 2 21:37 adresses.txt
−rw−−−−−−− 1 jice users 144 Aug 2 21:37 motsdepasse.txt
lrwxrwxrwx 1 jice users 14 Aug 2 21:37 linux −> /usr/src/linux
Explication :
La première ligne "total 144" est l'espace disque utilisé par l'ensemble des fichiers du répertoire.

1. La première colonne −rw−r−−r−−représente les **permissions** associées au fichier. le premier caractère est un tiret pour un fichier, un d pour un répertoire, un l pour un lien, etc.
   ensuite, on a trois groupes de trois caractères : rw− ou r−− ou rwx ou...
   Le premier groupe représente les permissions associées à l'utilisateur (ici, jice), le deuxième celles associées à son groupe (ici : users), enfin le dernier est les permissions que tout le monde a sur ces fichiers.
   r signifie : possibilité de lire ce fichier / dans ce répertoire,
   w signifie : possibilité d'écrire dans ce fichier / répertoire,
   x signifie : possibilité d'exécuter ce fichier / d'aller dans ce répertoire.
2. nombre d'inodes (partie élémentaire de [../docs/glossaire.php3#systeme_fichiers système de fichiers]) qui pointent vers le fichier/répertoire (généralement 1 pour un fichier, 2+le nombre de sous−répertoires pour un répertoire).
3. utilisateur à qui appartient le fichier (jice)
4. groupe auquel le fichier appartient (users)
5. taille en octets
6. date et heure de modification
7. nom du fichier/répertoire.

Quelques questions et réponses

Les commandes du tableau ci−dessus permettent de répondre à quelques questions comme :
Où est cette commande (which) ? Que contient ce fichier (cat ou tac) ? Quel fichier contient tel mot (grep) ? etc.
Voici d'autres questions et réponses qui ne nécessitent pas d'être root et vous permettront de continuer à vous familiariser avec bash et ses commandes. Elles sont regroupées en trois tableaux : Informations sur les commandes ; informations sur le système ; informations sur les fichiers.

Informations sur les commandes

Quelle commande utiliser pour faire ... ceci ou cela ?	apropos mot_clef ou man −k mot_clef − affiche les commandes, brièvement définies, en rapport avec mot_clef. apropos copier − affiche les commandes en rapport avec la copie d'un fichier, d'une chaîne, d'une zone mémoire ... apropos permission − affiche les commandes liées à la vérification et à la modification des permissions. Notes : les noms communs et les verbes à l'infinitif permettent généralement de trouver facilement la commande recherchée. En cas d'échec, pensez aux synonymes : apropos supprimer fait apparaître la commande rmdir (supprimer un répertoire), alors que apropos effacer fait apparaître la commande rm (effacer un fichier).
Comment se définit cette commande ?	whatis nom_commande ou man −f nom_commande − affiche une brève définition de nom_commande. whatis whatis − affiche la définition de whatis. whatis arch − affiche la définition de la commande arch. cd /bin ; for i in * ; do whatis $i ; done | more ; cd − se positionne dans le répertoire /bin/, affiche page par page la définition de chacune des commandes qui s'y trouve, retourne au répertoire personnel. Notes : dans l'exemple précédant vous pouvez bien sûr remplacer /bin par /sbin ; certaines entrées échouent.
Quelles sont et comment utiliser les commandes internes ?	help − affiche la liste des commandes internes et leur syntaxe. help nom_commande − affiche une aide sommaire sur nom_commande. help help − affiche une aide sur help. help alias − affiche une aide sur la commande alias. Notes : ces commandes sont les commandes internes du shell, généralement le bash. Vous pouvez aussi en obtenir la liste en demandant à propos de l'une d'elles une page man qui n'existe pas (man :). Vous trouverez également une aide sur les commandes internes dans la page man de bash (man bash). Enfin notez que help concerne les commandes internes mais −−help les commandes externes.
Où sont et quelles	ls /bin − affiche le contenu du répertoire /bin/, et donc la liste des

sont les commandes externes ?	commandes externes usuelles communes à tous les utilisateurs. ls /sbin − affiche le contenu du répertoire /sbin/, et donc la liste des commandes externes usuelles réservées à l'administrateur (root). whereis nom_commande − affiche le chemin de nom_commande ainsi que celui de sa page man. whereis cat − affiche le chemin de la commande cat (/bin/cat) et celui de sa page man (/usr/share/man/man1/cat.1.bz2). which nom_commande − affiche le chemin de nom_commande. which tac − affiche le chemin de tac (/usr/bin/tac). Notes : les répertoires /usr/bin/ et /usr/sbin/ contiennent les commandes externes moins fréquemment utilisées. Pour savoir si une commande est externe vous pouvez aussi simplement vérifier qu'elle n'est pas interne !
Comment obtenir un aide mémoire sur cette commande ?	nom_commande −−help − affiche l'aide mémoire de nom commande. ls −−help − affiche l'aide mémoire de la commande ls. Notes : −−help concerne la plupart des commandes externes et help les commandes internes. echo −−help affiche bien sûr ... −−help.
La dernière commande s'est−elle bien terminée ?	echo $? − affiche le code de retour de la dernière commande effectuée, 0 si elle s'est bien terminée, un autre nombre dans le cas contraire. clear ; echo $? − efface l'écran et affiche 0. sl / ; echo $? − affiche un message d'erreur et le code 127 (bash ne connaît pas la commande sl). nom_commande 2>/dev/null && echo "ok" || echo "m'enfin" − exécute nom_commande en redirigeant les erreurs vers /dev/null (périphérique fictif) puis affiche "ok" si tout s'est bien passé ou "m'enfin" dans le cas contraire. : && echo "ok" || echo "m'enfin" − ne fait rien puis affiche "ok" (la commande : ne fait rien et se termine toujours bien). bof 2>/dev/null && echo "ok" || echo "m'enfin" − affiche "m'enfin" (la commande bof n'existe pas). Notes : commande1 && commande2 exécute commande2 que si commande1 s'est terminée normalement (0) ; commande1 || commande2 exécute commande2 que si commande1 ne s'est pas terminée correctement (<>0).

Quels sont le nom, la taille et le contenu du fichier d'historique ?	echo $HISTFILE $HISTFILESIZE
	  − affiche le nom et la taille du fichier d'historique des commandes.
	cat $HISTFILE | more
	  − affiche le contenu du fichier d'historique page par page.
	tail −n 24 $HISTFILE
	  − affiche les 24 dernières lignes du fichier d'historique.
	
	Notes : le fichier d'historique vous aide à répondre à la question : mais comment j'avais fait ? et à en conserver une trace d'une connexion à l'autre.

Quel est l'historique actuel ?	history | more
	  − affiche page par page le contenu numéroté de l'historique actuel des commandes.
	history 12
	  − affiche les 12 dernières entrées effectuées et leurs numéros.
	
	Notes : c'est cet historique qui défile avec les touches "flèches" de votre clavier ; au démarrage, c'est le contenu de votre fichier d'historique.

Informations sur le système

Quel est le système ?	uname ou echo $OSTYPE
	  − affiche le nom du système d'exploitation.
	uname −a
	  − affiche diverses informations système (nom du SE, version, microprocesseur ...).
	arch ou uname −m
	  − affiche le type du microprocesseur.
	cat /proc/cpuinfo
	  − affiche des informations sur le microprocesseur (type, fréquence, cache ...).
	
	Notes : les informations recueillies avec uname −a peuvent aussi être obtenues avec certaines variables système : echo $OSTYPE $BASH $BASH_VERSION etc.

Depuis combien de temps ce shell est−il actif ?	uptime
	  − affiche l'heure, la durée d'activité du système, le nombre d'utilisateurs ...
	echo $SECONDS
	  − affiche la durée d'activité du shell courant en secondes.
	echo $[$SECONDS/3600] h $[($SECONDS%3600)/60] mn $[$SECONDS%60] s
	  − affiche cette durée en heures, minutes et secondes.
	
	Notes : bash ne connaît que les entiers et donc que les divisions euclidiennes, / permet d'en obtenir le quotient et % d'en obtenir le reste (ou modulo). Chaque fois que vous changez de terminal, ouvrez un nouvel xterm ou faites un su, celui−ci correspond à un nouveau (sous)shell, sa variable SECONDS est donc alors à 0. Pour savoir depuis combien de temps vous êtes sous Linux, revenez à votre console de login ; pour savoir depuis quand le système est actif, utilisez uptime.

Qu'en est−il des disques, de la	mount − affiche la liste des disques montés.

mémoire, du microprocesseur ?	df −ah
	  − affiche au format humain l'espace total, occupé, libre sur tous les disques.
	free ou cat /proc/meminfo
	  − affiche des informations sur la mémoire (totale, libre, swap ...).
	vmstat
	  − affiche les statistiques sur la mémoire virtuelle.
	top
	  − affiche et permet d'observer en temps réel l'activité de la mémoire et du microprocesseur.
	
	Notes : la touche <Q> permet de quitter la commande top.

Quelles sont et que signifient les variables système ?	$<Tab>
	  − affiche les noms des principales variables système.
	help variables | more
	  − affiche page par page la définition des principales variables système.
	
	Notes : les noms des variables système sont généralement écrits en majuscules. A chacune de ces variables correspond une question et une réponse, quelques unes seulement ont été formulées à divers endroits dans cette page.

Que contiennent ces variables système ?	echo $NOM_VARIABLE
	  − affiche le contenu de NOM_VARIABLE.
	echo $USER
	  − affiche le nom de l'utilisateur en cours.
	echo $PS1
	  − affiche la chaîne de caractères définissant le prompt.
	printenv | more
	  − affiche page par page l'environnement.
	set | more
	  − affiche page par page le contenu des principales variables système.
	
	Notes : vous pouvez bien sûr modifier le prompt PS1="chaîne de caractères". Pour rendre cette modification générale et durable, il faut utiliser le fichier /etc/bashrc (nécessite d'être root).

Qui suis−je et qui est ou était connecté au système ?	id nom_utilisateur
	  − affiche les identifications de nom_utilisateur.
	id
	  − affiche vos identifications : UID, GDI, groupes.
	who am i
	  − affiche vos coordonnées dans le système.
	who −H
	  − affiche avec une entête des informations (nom de login, console ...) sur les utilisateurs connectés.
	w
	  − affiche qui est connecté (who −H) et aussi ce qu'il fait.
	last −n 12
	  − affiche la liste des 12 dernières connexions.

	Notes : la commande who am i ne fonctionne pas toujours sous xterm, dans ce cas entrez whoami mais vous n'obtiendrez que votre nom de login. De nombreuses commandes permettent d'obtenir une partie des renseignements évoqués ci−dessus, leurs noms parlent d'eux−mêmes : logname, users, groups ...
Quelles sont la date et l'heure ?	date − affiche la date et l'heure. br />date +%x − affiche seulement la date au format jj.mm.aaaa. date +%X − affiche uniquement l'heure au format hh:mm:ss. cal − affiche un calendrier du mois en cours. cal 2000 − affiche le calendrier de l'an 2000, etc.
Y a−t−il un pingouin dans le système ?	linux_logo − affiche Tux et diverses informations système. linux_logo −la − affiche Tux en ASCII et sans informations :−) Notes : sur mon système, linux_logo −la produit bien l'affichage attendu (−a = ASCII et −l = pas d'informations) mais avec linux_logo −al, l'option −l est ignorée et les informations sont affichées ... Y a−t−il un bug dans mon système ?

Informations sur les fichiers

Quel est le répertoire courant ?	pwd − affiche le nom complet du répertoire en cours.
Quelle est la taille de ce répertoire ?	du −sh − affiche au format humain la taille du répertoire courant. du −h ~ | more − affiche page par page au format humain la taille de votre répertoire personnel et de tous ses (sous)répertoires. Notes : du ne fonctionne que sur les répertoires où vous avez droit d'accès, si nécessaire passez sous root.
Quel est le type de ce fichier ?	file nom_fichier − affiche le type de nom_fichier. file * − affiche le type de tous les fichiers du répertoire en cours. file /bin − affiche que /bin/ est un répertoire. file /bin/sh − affiche que sh est un lien symbolique vers bash. file /bin/bash − affiche que bash est un exécutable. file ~/.bashrc − affiche que .bashrc de votre répertoire personnel est un fichier texte. file /dev/null

	− affiche que null est un fichier spécial. Notes : file sait reconnaître un grand nombre de types de fichiers parmi les fichiers spéciaux, exécutables, textes, données ... Il est préférable de l'utiliser avant d'entrer un "cat" au hasard.

Raccourcis

Clavier

- **Tab**

Taper une fois la touche [Tab] permet de compléter automatiquement un nom de fichier/répertoire s'il est unique :

[user@localhost user]$ cd /et**[Tab]**
[user@localhost user]$ cd /etc_

- **Tab Tab**

Si lors du premier appui sur [Tab], le nom n'a pas été complété, un deuxième appui vous donne la liste de toues les possibilités :

[user@localhost user]$ cd /usr/doc/H**[Tab][Tab]**
HTML HOWTO
[user@localhost user]$ cd /usr/doc/H_

- **Flèche vers le haut (ou Ctrl−P) / bas (ou Ctrl−N)**

La flèche vers le haut permet de remonter dans l'historique des commandes, la flèche vers le bas permet de revenir. Vous pouvez aussi utiliser la commande fc, consultez man fc.

- **Shift − flèche vers le haut/bas**

Permet de scroller le contenu du terminal texte vers le haut ou le bas, d'une ligne. En effet, les lignes qui ont défilé vers le haut restent en mémoire et restent accessibles. Terrible, non ?

- **Shift − Page up/down**

La même chose, mais page par page.

- **Ctrl−C**

Arrête le processus en cours, celui qui a été lancé par la dernière commande.

- **Ctrl−Z**

Stoppe le processus en cours, celui qui a été lancé par la dernière commande, mais ne le détruit pas : il reste en attente. Pour le mettre en tâche de fond (il continue à s'exécuter, mais vous pouvez continuer à taper des

commandes), tapez **bg**. Pour le faire revenir en avant, taper **fg**.

- **Ctrl−D**

Ferme le terminal en cours (similaire à exit ou logout).

et aussi...

- **Ctrl−Alt−Fn**

Se place sur la console virtuelle numéro **n**. Par défaut, il y a en général 6 consoles texte virtuelles, de F1 à F6, et X Window se lance dans la septième (F7).

- **Ctrl−Alt−Backspace**

Cela permet de tuer X et de revenir soit à la [../docs/glossaire.php3#display_manager bannière de login] soit au shell qui a lancé X par startx. A éviter si possible : il est plus sain de quitter X en se déconnectant proprement.

- **Ctrl−Alt−Del**

Suivant votre configuration, ces touches à l'action bien connue vous permettrons de rebooter votre ordinateur (synonyme de reboot ou shutdown −r now). Si vous souhaitez juste arrêter votre ordinateur afin de l'éteindre, tapez halt, ou shutdown −h now.

Shell

Les shells Unix disposent de petits "raccourcis" très astucieux et utiles, qui vous épargnerons de taper sur quelques touches. Ne dit−on pas qu'un bon informaticien est un informaticien fainéant ? :−)

Dernière ligne de commande : !!

On a vu plus haut qu'elle était accessible par la flèche vers le haut, mais vous pouvez également la désigner par '!!', ce qui peut être très intéressant.

[user@localhost user]$ vi
[user@localhost user]$ which !!
which vi
/bin/vi
Arguments de la dernière commande : !*

Les arguments de la dernière commande peuvent être représentés par '!*'.

[user@localhost user]$ mkdir test
[user@localhost user]$ cd !
cd test
Utiliser la sortie d'une commande comme argument :

Vous pouvez réutiliser directement ce qu'une commande écrit à l'écran comme argument pour une autre commande. Pour ce faire, vous devez encadrer la commande par une cote inverse ` ou la mettre entre

parenthèses précédées du signe $; elle sera remplacée par ce qu'elle écrit à l'écran dans la ligne de commande. Imaginez par exemple que vous vouliez voir les informations sur le fichier exécutable de emacs.

[user@localhost user]$ ls −l `which emacs`
[user@localhost user]$ ls −l $(which emacs)
est ainsi équivalent à :

[user@localhost user]$ which emacs
/usr/bin/emacs
[user@localhost user]$ ls −l /usr/bin/emacs
Cool non ? Et vous pouvez mixer les raccourcis vus précédemment :

[user@localhost user]$ emacs
[user@localhost user]$ ls −l `which !!`
C'est−y pas beau ça madame ?

Remplacer un caractère par un autre : ^

Si vous souhaitez remplacer la première occurrence d'un caractère de la ligne de commande précédente par un autre, vous pouvez utiliser le symbole ^, comme ci−dessous :

[user@localhost user]$ lpcate i486−linux−libc5
lpcate : command not found
[user@localhost user]$ ^p^o
locate i486−linux−libc5
^p^o signifie : refait la même ligne de commande que précédemment, mais remplace le premier p par un o.

Lancer un programme directement en tâche de fond : &

Il suffit de faire suivre la ligne de commande du symbole & :

[user@localhost user]$ cp −R /usr/doc /tmp &
[1] 7194
[user@localhost user]$ _
La commande est lancée en tâche de fond, c'est à dire qu'elle s'exécute, mais la main vous est rendue tout de suite. La fin de la commande est signifiée par un message :

[user@localhost user]$
[1]+ Done cp −R /usr/doc /tmp
[user@localhost user]$ _
Lancer plusieurs programmes en même temps : &, &&, ||, ;

Vous avez plusieurs solutions :

prog1 ; prog2 lance prog1, puis prog2,

prog1 & prog2 lance prog1 en arrière plan, puis immédiatement prog2 en avant plan,

prog1 && prog2 lance prog1, puis prog2 seulement si prog1 n'a pas retourné d'erreur,

prog1 || prog2 lance prog1, puis prog2 seulement si prog1 A retourné une erreur.

Redirections

Normalement, la sortie des programmes se fait à l'écran, aussi bien pour les erreurs (*standard error*) que pour les messages "normaux" (*standard output*). Vous pouvez la rediriger, soit vers un fichier avec >, soit vers l'entrée d'un autre programme avec | (ou pipe − attention, arrêtez de rigoler dans le fond :−).

De même, l'entrée standard (*standard input*) est habituellement constituée du clavier, mais on peut aussi la remplacer par le contenu d'un fichier, avec le symbole <.

Envoyer la sortie standard d'un programme dans l'entrée standard d'un autre

Vous avez déjà certainement vu ou utilisé une commande du type :

[user@localhost user]$ ls −la | more
ls −la envoie la version longue du listing de répertoire, avec les fichier cachés, à more qui l'affiche page par page.
Vous pouvez aussi enchaîner plusieurs redirections :

[user@localhost user]$ cat /var/log/messages | grep gpm | more
Ceci va afficher page par page l'ensemble des messages système relatifs à gpm. (voir plus haut ce que font chacune de ces commandes).

Envoi d'un fichier dans l'entrée standard

L'entrée standard (*standard input*) est normalement ce que vous tapez au clavier. Vous pouvez remplacer vos frappes clavier par le contenu d'un fichier, qui sera ouvert et envoyé sur l'entrée standard du programme. C'est pratique pour automatiser des tâches avec des programmes interactifs. Exemple :

[user@localhost user]$ ftp < sessiontype.txt
Ici le fichier sessiontype.txt pourra contenir par exemple :

```
open ftp.lesite.com
user jice
pass xxxxxx
cd /pub/linux/doc
bin
get jice.jpg
bye
```

Et vous permettra en une seule commande de récupérer le fichier jice.jpg sur le site ftp.lesite.com (utile si ce fichier change et que vous voulez le mettre à jour régulièrement). Bref, à vous d'inventer la vie qui va avec :−)

Redirection des sorties vers un fichier

[user@localhost user]$ ls −lR /cdrom > cdrom.txt
Cette commande va lister le contenu du cdrom, et enregistrer le résultat dans le fichier cdrom.txt.
En mettant deux > de suite, vous ajoutez au fichier :

[user@localhost user]$ date **>>** cdrom.txt
Ceci va ajouter la date au fichier précédemment créé.

Les messages d'erreur peuvent être dirigés séparément dans un fichier avec **2>** :

[user@localhost user]$ startx > startx.log **2>** startx.err
ou dirigés vers le même fichier que les messages normaux :

[user@localhost user]$ startx > startx.log **2>&1**

Gestion des processus

Linux est *multitâches*, ce qui signifie que plusieurs programmes (qui peuvent être à la fois des applications utilisateur ou des tâches système) peuvent tourner simultanément. On vient de voir qu'on pouvait lancer directement depuis un terminal texte une commande en tâche de fond, avec le symbole &.
Comment gère−t−on ensuite ces processus ?

Lister les processus : ps

La liste des processus en cours pour un terminal donné s'obtient en tapant simplement la commande **ps** :

```
 PID TTY STAT TIME COMMAND
  12  p1 S    0:00 bash
 144  p1 S    0:01 emacs
1768  p1 R    0:00 ps
```

Si vous voulez voir plus de processus, vous pouvez lister tous les processus d'un utilisateur par ps U root :

```
  PID TTY      STAT   TIME COMMAND
    1 ?        S      0:04 init
    2 ?        SW     0:00 [keventd]
    3 ?        SWN    0:00 [ksoftirqd_CPU0]
...
```

Vous pouvez aussi voir l'ensemble des process d'un système par ps aux

Lister les jobs et les gérer : jobs, fg, bg

"job" est un mot qui désigne ici les programmes que vous avez lancé en arrière plan (tâche de fond) dans votre terminal. Pour lancer un job en arrière plan, vous pouvez :

lancer le programme par son nom, puis taper Ctrl−Z pour le stopper, puis la commande **bg** pour l'envoyer en arrière plan (BackGround).

vous pouvez aussi simplement taper le nom de ce programme suivi par le symbole **&**.

Afin d'afficher une liste des jobs d'un terminal, tapez la commande **jobs**:

[user@localhost user]$ find / −name "*a*" >A &
[1] 7859
[user@localhost user]$ jobs
[1]+ Running find / −name "*a*" >A &

Pour chacun de ces jobs, vous pouvez les faire revenir en avant plan avec la commande **fg** ; "fg" pour le dernier programme lancé en tâche de fond, "fg %n" pour le n^{ième}.

Tuer un processus : kill, killall

Afin de terminer un processus qui ne répond plus, par exemple, on utilise la commande **kill**, suivie du numéro de job (%n) ou du PID du programme à tuer. Par exemple, si ps donne le résultat ci−dessus, la commande "kill 144" arrêtera la tâche emacs. "kill %1" fera la même chose.

Vous pouvez également tuer des processus par leur nom avec la commande **killall** suivie du nom du processus à tuer, mais attention : TOUS les processus de l'utilisateur utilisant killall et portant le même nom seront tués. Par exemple, si vous tapez "killall emacs", non seulement la fenêtre emacs lancée depuis ce terminal sera supprimée, mais aussi tous les autres emacs lancés depuis un autre terminal par l'utilisateur.

Vous pouvez aussi passer un autre argument à kill et killall, qui est le *signal* à envoyer à la tâche (les *signaux* sont une manière de communiquer avec les applications sous Unix). Par exemple, si la tâche récalcitrante ne s'arrête pas avec un simple kill 144, essayez kill −9 144, ou kill −QUIT 144.

Aliases et variables d'environnement

Aliases

Plutôt que de taper de longues commandes, ou bien parce que vous préférez vous rappeler d'un nom plutôt que du vrai nom Unix, vous pouvez définir des *aliases*. Pour ce faire, utilisez la commande alias comme suit :

Si votre shell est **bash** ou sh ou ash (par défaut) :

alias md=mkdir
alias ls='ls −−color'
alias eclip2='telnet eclip2.ec−lille.fr'
Si votre shell est **tcsh** ou csh (par défaut) :

alias md mkdir
alias ls 'ls −−color'
alias eclip2 'telnet eclip2.ec−lille.fr'
Ainsi pourrez−vous taper md au lieu de mkdir, et eclip2 pour vous connecter à cette machine via telnet ; la commande ls affichera une sortie en couleurs...

Le problème est que les aliases définis dans un terminal ne sont valables que dans celui−ci, et disparaîtrons à jamais dès que ce terminal sera fermé. Pour conserver des alias par−delà les connexions/déconnexions, regardez la [#configuration_shell configuration du shell] : vous pouvez définir vos aliases dans le fichier **~/.bashrc**.

Variables d'environnement, Path et Prompt

Les variables d'environnement servent à enregistrer des paramètres que les programmes peuvent lire ensuite. Elles sont désignées par un symbole $ suivi de lettres, chiffres et symboles.

Par exemple, la variable $HOME est égale au répertoire maison de l'utilisateur en cours (en général

/home/user).

De même, la variable $PATH représente le chemin de recherche que le shell va parcourir afin de trouver le fichier exécutable qui correspond à la commande que vous venez de taper. Par exemple, $PATH = /bin:/usr/bin:/usr/local/bin.

Créer ou modifier une variable d'environnement
Si votre shell est **bash** ou sh ou ash (par défaut) :

export MAVARIABLE=mavaleur
Si votre shell est **tcsh** ou csh (par défaut) :

setenv MAVARIABLE mavaleur
Cette commande positionnera la variable MAVARIABLE à la valeur mavaleur. Vous pouvez le vérifier, en tapant la commande echo $MAVARIABLE qui écrira à l'écran "mavaleur".

Vous pouvez ainsi ajouter le chemin /home/user/bin à votre $PATH si vous installez des logiciels dans votre répertoire personnel par exemple. Sous bash, cela donnera : export PATH=$PATH:/home/user/bin. Cependant, de même que pour les aliases, ce nouveau PATH sera perdu dès votre déconnexion...

Attention, en général, le répertoire courant '.' ne fait pas partie du PATH pour des raisons de sécurité : imaginez qu'une personne mal intentionnée aie mis un programme destructeur appelé "ls" dans votre répertoire, vous le lanceriez dès que vous taperiez la commande ls ! C'est pourquoi il faut toujours faire précéder de son chemin complet une commande qui n'est pas dans le PATH, et ce même si vous êtes dans le même répertoire que la commande ! Ainsi, il ne faut pas taper configure, mais ./configure (programme classique à lancer avant compilation d'un logiciel), ce qui signifie : lance le programme 'configure' qui est présent dans le répertoire courant.

Le **prompt** est également contenu dans une variable d'environnement : PS1.
Le prompt par défaut de la Mandrake par exemple, [user@localhost user]$, est défini comme suit : PS1="[u@h W]$ ".
u est l'utilisateur, h le nom de machine (hostname), w le chemin courant (ex : /usr/doc), W le répertoire courant (ex : doc)... voyez man bash pour l'ensemble des possibilités.

Une autre variable d'environnement utile : PROMPT_COMMAND. Cette variable contient une commande qui est exécutée à chaque fois que le prompt est affiché. Cela permet des tas de fantaisies rigolotes, comme par exemple de jouer un son (trop utile :−) ou de positionner le titre d'un xterm avec le nom du répertoire courant (voir man xterm).

Regardez la [#configuration_shell configuration du shell] : vous pouvez définir vos variables dans le fichier ~/.bash_profile.

Configuration du shell

Vous pouvez enregistrer des fichiers qui seront lus et exécutés par votre shell, lors de l'ouverture d'un terminal, aussi bien que lors de sa fermeture. Cela va vous permettre d'y placer vos aliases préférés, et vos variables d'environnement.

Pour **bash** et consorts, ces fichiers s'appellent : .bashrc, .bash_profile pour la connexion et .bash_logout pour la déconnexion.

Pour **tcsh** et ses potes, ces fichiers s'appellent : .tcshrc, .login pour la connexion et .logout pour la déconnexion.
Ces fichiers se situent tous dans le répertoire maison de l'utilisateur ($HOME). Notez bien qu'ils commencent par un point : ce sont des fichiers cachés. Pour les voir, il faut faire un "ls −a".

Examinez les avec votre éditeur de texte préféré, et vous verrez comment ajouter de nouveaux aliases et variables d'environnement, ainsi que lancer tel ou tel programme automatiquement : en tant que *scripts shell*, ces programmes sont en fait une suite d'instructions qui sera interprétée par le shell.

Les entrailles du shell

Vous êtes maintenant munis d'une jolie batterie d'outils qui va entre autres vous permettre d'écrire tous vos scripts shell. Toutefois même en ayant récupéré la synthaxe de commandes vous continuez à subir des erreurs... Soit la commande ne vous retourne pas le résultat attendu soit le shell vous retourne des erreurs... Ne vous êtes vous jamais posé cette question : est−ce que je mets des simples quotes, des doubles quotes ou des back quotes ?
Pour ne plus avoir à se poser ce genre de questions, il est essentiel de bien connaitre le fonctionnement interne du shell.

Les grandes étapes de l'interprétation d'une ligne de commandes

Pour mieux comprendre le résultat obtenu, il faut savoir que le shell lit plusieurs fois la ligne avant d'exécuter la commande.
Cette lecture se fait dans l'ordre suivant :

1. substition de variables : le shell remplace les variables par leurs valeurs
2. substitution de commandes : le shell remplace une variable par son contenu qui est le résultat d'une commande
3. interprétation des pipes et des redirections
4. expansion des noms de fichiers : interprétation des caractères spéciaux pour compléter un nom de fichier et/ou de répertoire
5. exécution de la commande

On se rend donc compte qu'un caractère spécial peut être interprété par le shell avant d'être interprété au sein de la commande.

Exemple : le caractère "*" peut être interprété par le shell (remplace 0 ou n caractères pour compléter un nom de fichier) ou par une commande comme grep (répète de 0 à n fois le caractère précédent dans une chaîne de caractères). Toutefois, sans précision dans la synthaxe, et selon les étapes ci−dessus, le caractère sera d'abord interprété par le shell. D'où quelques surprises dans le résultat de la commande.

L'interprétation des caractères spéciaux

Pour choisir de faire interpréter les caractères spéciaux par le shell ou la commande, il existe 3 possibilités :

- **utilisation des doubles quotes** (ou guillemets en bon français) : lorsque la chaîne de caractères est écrite entre guillemets, tous les caractères spéciaux perdent leur signification **sauf** : **$** \ et `` (simples quotes).

exemple :

[user@localhost user]$ echo *
bidule fic1 fic2 truc
***** est un caractère spécial non protégé. Au premier passage du shell, il est donc interprété. Il signifie alors l'ensemble des fichiers du répertoire courant

[user@localhost user]$ echo "*"
*
***** est protégé par les guillemets. Il n'est donc pas interprété par le shell comme caractère spécial et devient un caractère pour la commande echo. Dans le cadre de cette commande, * n'a aucune signification particulière, il est donc affiché à l'écran tel que.

- **utilisation des simples quotes** : lorsque la chaîne de caractères est écrite entre simples quotes, tous les caractères spéciaux sans exception perdent leur signification pour le shell. Ils seront donc éventuellement interprété par la commande passée.

exemple : recherche des lignes d'un script utilisant la variable PATH

[user@localhost user]$ grep $PATH /home/user/script
[user@localhost user]$
Au premier passage, le shell interprète le **$** comme introduisant une variable. Il remplace donc la variable par son contenu puis exécute la commande grep. Il ne trouve donc aucune ligne comportant les chemins référencés dans PATH.

**[user@localhost user]$** grep '$PATH' /home/user/script
echo $PATH
[user@localhost user]$
Le shell n'interprète pas le **$** lors du premier passage car sa signification est annulée par les simples quotes. le $ est donc traité par la commande grep. Comme il n'a pas de signification particulière, il est interprété comme un caractère quelconque.

- annulation d'un caractère spécial avec \ : pour empêcher le shell d'interpréter un caractère spécial, il suffit de positionner un anti−slash devant le caractère spécial donné.

exemple : pour reprendre l'exemple précédent on aurait pu écrire aussi :

[user@localhost user]$ grep \$PATH /home/user/script
echo $PATH
[user@localhost user]$

Liste des caractères spéciaux

Ci−dessous la liste des caractères spéciaux du shell :

&	processus en arrière−plan
~	home directory
;	séparateur de commandes
\	annulation d'un caractère spécial

"	doubles quotes : encadre une chaîne de caractères et annule la signification de $, \ et '
`	back quotes : substitution de commandes
"	simples quotes : encadre une chaîne de caractères et annule la signification de tous les caractères spéciaux
#	commentaire
( )	exécution d'un shell fils
[ ]	test
|	pipe
$	variable
*****	remplace 0 ou n caractères
!	négation d'un test
?	remplace 1 caractère
< >	redirections entrée, sortie
$0...$9	variables de position

Appel des commandes

Autre élément à connaître pour ne pas avoir de surprise : à quel type de commande ai−je à faire ? on distingue des grands types de commande

- des commandes internes au shell : elles sont exécutées dans le même shell. si elles sont lancées à partir du shell père, il n'y aura pas création de shell fils
- des programmes binaires exécutables
- des fichiers de commande (shell−scripts)
- des alias
- des fonctions

La difficulté c'est que le shell interprète une commande en suivant un ordre très précis :

- **le chemin de la commande comporte un /**, il exécute donc la commande située dans ce chemin, il n'y a pas d'ambiguïté possible.
- **le chemin de la commande ne comporte pas de /**, il cherche la commande en suivant les étapes suivantes :
    1. consultation de la liste des alias
    2. consultations des fonctions chargées
    3. consultations des commandes internes du shell
    4. consultation de la variable PATH

Donc si vous tapez une commande quelleconque, un script shell par exemple, sans préciser le chemin, la consultation de la variable PATH n'arrive qu'en dernier. Attention si vous disposez d'un alias ou d'une fonction qui porte le même nom, il traitera l'alias ou la fonction.

Pour terminer 2 commandes utiles pour savoir à quel type de commande vous avez à faire :

- "'type nom_de_commande"' : permet de déterminer le type d'une commande (alias, fonction, commande interne...)
- "'which nom_de_commande"' : recherche le chemin de la commande dans PATH

exemples :

[user@localhost user]$ type ls
ls est un alias suivi pour /usr/bin/ls
[user@localhost user]$ type cd
cd est une commande prédéfinie du shell
[user@localhost user]$ which pwd
/usr/bin/pwd

Index des commandes

heu... bon là j'en ai marre, on verra ça plus tard ! :−)

Je place quand même ici 2 commandes qui peuvent être bien utiles :

cal : donne le calendrier du mois courant,
cal 12 1999 : donne le calendrier de décembre 1999,
cal 2000 : donne le calendrier des 12 mois de l'an 2000.

"'factor"' 12456988 : donne la décomposition en produit de facteurs premiers du nombre 12456988 (soit 2 x 2 x 17 x 183191) − c'est très mathématique, mais ultra rapide et puissant.

Pour terminer cet article, je vous renvoie vers ce petit manuel de référence, qui contient l'ensemble des commandes usuelles :

-