## Administration système (résumé)

# L’arborescence sous Linux

Sous Linux, tout est fichier !

Chacun de ces fichiers est placé quelque part en dessous de la racine / (« root »)

Dans /, il faut un ensemble de répertoires systèmes dont la présence de certains est impérative (\*), et pour d'autre, préférable.

* **/dev** : C'est ici que les périphériques (réels et virtuels) sont accessibles (partitions, disques, cartes son, ports SCSI, ports USB, etc.), mais en mode "données brutes". Il est souvent nécessaire d'utiliser des programmes pour interpréter ces contenus (par exemple, en montant la partition /dev/hda1 dans /mnt/hda1 pour accéder aux fichiers).
* **/mnt** : C'est en général à cet endroit qu'on accède aux autres systèmes de fichiers (autres partitions, CD/DVD, clés USB, serveurs de fichiers...)
* **/etc**(\*) : Ici sont regroupés tous les fichiers de configurations des différents logiciels installés sur la machine ainsi que des fichiers de configuration système utilisés au démarrage de la machine.
* **/media** : Certaines distributions montent les périphériques amovibles à cet endroit.
* **/var** : Fichiers dont le contenu varie
	+ **/var/log** : On trouve ici les logs des différents logiciels et serveurs. Cela permet de voir ce qui s'est passé quand quelque chose ne va pas.
	+ **/var/spool** : Fichiers en cours de traitement (file d'impression, mails en cours d'envoie...)
	+ **/var/tmp** : Fichiers temporaires (voir aussi /tmp).
* **/home** : Chaque utilisateur possède son propre répertoire pour y stocker ses fichiers personnels et la configuration des programmes.
* **/usr**(\*) : Répertoire contenant les fichiers du système partageables en réseau et en lecture seule.
* **/opt** : Répertoire contenant les applications complémentaires (dites : add-on) n’appartenant pas à la distribution installée.
* **/tmp** : Sont stockés ici les fichiers temporaires (fichiers créés pendant le fonctionnement des logiciels et supprimés à la fin). (Voir aussi /var/tmp)
* **/boot**(\*) : Sont stockés ici les fichiers de démarrage du système (noyau du système, etc.). On y trouve aussi certains fichiers de configuration (GRUB...)
* **/lib**(\*) : Ce sont des bibliothèques utilisées par divers programme (C'est l'équivalent des DLL Windows). Par exemple, libjpeg.so permet à tous les programme de lire et écrire des fichiers JPEG.
* **/sbin**(\*) : Ce répertoire contient les programmes systèmes et les outils d'administration (par exemple les outils permettant de formater un disque).

**/bin**(\*) : Ici sont situés les programmes utilisés à la fois par les utilisateurs et les administrateurs. Ensemble de fichiers exécutables représentant les commandes.

# L'environnement shell

Le shell met à sa disposition des variables d'environnement, c'est-à-dire un conteneur mémoire dans lequel des données sont stockées.
Pour afficher le contenu d'une variable d'environnement, la commande echo $NOM\_VARIABLE peut être utilisée.

Le nom des variables d'environnement est par convention en majuscules, il est donc nécessaire de respecter la casse.

# Variables d'environnement :

HOME, USER, GROUPS, UID, PWD, SHELL, PATH, HOSTNAME

* HOME contient le répertoire d'utilisateur
* USER contient le login d'utilisateur
* PWD contient le répertoire courant
* SHELL contient le nom du shell de connexion
* PATH contient la liste des répertoires où se trouvent les commandes que l'utilisateur peut exécuter
* HOSTNAME contient le nom de la machine
* HISTSIZE contient la taille maximale des commandes exécutées contenues dans le fichier historique
* PS1 contient les paramètres d'affichage de l'invite de commande (le prompt)

# Manipuler des variables

Après connexion, l'utilisateur est connecté dans son environnement. Cela signifie que le shell met à sa disposition des variables d'environnement, c'est-à-dire un conteneur mémoire dans lequel des données propre à chaque utilisateur ou au système sont stockées.
Pour afficher le contenu d'une variable d'environnement, la commande **echo $NOM\_VARIABLE** peut être utilisée.

Le nom des variables d'environnement est par convention en majuscules, il est donc nécessaire de respecter la case pour ces variables.

Exemples:

Donne l’invite de commande du shell :

$ echo $PS1

# Variables locales

Dans cette section, on va voir un aperçu sur les variables locales.

Ici, nous effectuerons juste un affichage « Hello world » et l’addition de 2 variables A et B :

Exemples:

Pour faire afficher au terminal sur l’écran la chaîne « Hello world » au travers d’une variable locale :

$ var=’Hello world’

$ echo bonjour, $var

Maintenant le calcul de A + B :

$ A=1 ;B=2

$ echo “$A+$B”

Il ne donne pas ce à quoi l’on s’attend… normalement, bash traite les valeurs des variables comme des chaînes de caractères. On peut effectuer des calculs sur des nombres entiers, en utilisant la syntaxe $(( ... )) pour délimiter les expressions arithmétiques:

$ echo $(($A+$B))

# Fichiers de configuration

Au moment de la connexion, dans une console virtuelle ou à l'ouverture d'un terminal en mode graphique, le shell utilise des informations qui se trouvent dans certains fichiers (.bashrc, .bash\_profile, etc).  Le comportement du shell peut être modifié en éditant ces fichiers.

# Où se trouvent les commandes ?

Les commandes que vous pouvez exécuter depuis votre terminal se trouvent dans certains répertoires de votre système.

La variable [PATH](http://www.commentcamarche.net/faq/3585-bash-la-variable-d-environnement-path) (en français : "chemin") contient une liste de répertoires qui contiennent les commandes accessibles. Pour avoir accès à toutes les commandes il faut généralement être [root](http://fr.wikipedia.org/wiki/Utilisateur_root).

Pour trouver l'emplacement d'une commande, on utilise "whereis", ou encore "which"

# Commandes de base

cat - Lit (concatène) un ou plusieurs fichier(s), affichage sur la sortie standard

cd - ChangeDirectory, change de répertoire

chmod - CHangeMODe - change le mode d'accès (permissions d'accès) d'un ou plusieurs fichier(s)

chown - CHangeOWNer - change le propriétaire d'un ou de plusieurs fichier(s)

cp - copier des fichiers

crontab - planification de tâches

cut - Retire des parties précises de texte dans chaque ligne d'un fichier

date -Affiche la date selon le format demandé

dd - DevicetoDevice - Recopie octet par octet tout ou partie du contenu d'un périphérique (habituellement de stockage) vers un autre péripherique.

df - affichage de la quantité d'espace libre disponible sur tous les systèmes de fichiers

du - DiksUsage - l'utilisation de disque

echo - Affiche du texte sur la sortie standard (à l'écran)

exit - arrête l'exécution du shell

find - recherche de fichiers

fsck - FileSystemChecK - vérification d'intégralité de système de fichiers

grep - recherche dans un ou plusieurs fichiers les lignes qui correspondent à un motif

groupadd- Ajouter un groupe d'utilisateurs

gunzip - décompression de fichiers

gzip - compression de fichiers

head - affiche les premières lignes (par défaut 10) d'un fichier

help - affiche une aide sur les commandes internes de bash

kill - envoyer un signal à un processus

less - programme d'affichage à l'écran

ln - création de liens

ls - liste le contenu des répertoires

man - affiche les pages de manuel

mkdir - MaKeDIRectory - crée un répertoire

mkfs - MaKeFileSystem - création de systèmes de fichiers

more - programme d'affichage à l'écran

mount - monter un système de fichiers

mv - déplacer, renommer un fichier

ps - affiche les processus en cours d'exécution

pwd - Print name of current/working directory - affiche le chemin complet du repertoire courant

rm - suppression de fichiers

rmdir -Remove empty directories - suppression d'un dossier vide

tail - affiche les 10 dernières lignes d'un fichier

tar - création d'archives

su - Substitute User identity ou Switch User - prendre l'identité d'un utilisateur

uname - Affiche des informations sur le système.

useradd - ajouter un utilisateur

whereis - localiser une commande

# Exécution d'une commande

Il y a plusieurs façons d'exécuter une commande.

* en utilisant tout simplement son nom
* en utilisant le chemin absolu
* en utilisant le chemin relatif
* utilisation des alias (pratique pour les commandes employées souvent et qui sont longues)

Une commande peut être exécutée en arrière plan en utilisant l'esperluette (&) après le nom de la commande.
L'exécution d'une commande en arrière plan permet de redonner la main au shell après l'exécution.

# Les alias

L'utilisation des alias est très pratique pour les commandes longues qui sont utilisées régulièrement. Ca évite de les retaper.
L'utilisation excessive des alias peut vous faire oublier les commandes et leurs options.
A vous de gérer l'utilisation des alias.

Les alias on les écrit dans le fichier .bashrc de la manière suivante

alias nom='commande'

Une fois le fichier /home/user/.bashrc édité, tapez

source /home/user/.bashrc

pour prendre en compte immédiatement les alias.

La commande alias affiche les alias existants.

# Les raccourcis clavier

**cd** : revenir dans le répertoire personnel

**cd -** : revenir dans le répertoire précédent (uniquement si vous avez exécuter un cd)

**Ctrl+l** : effacer l'écran

**Ctrl+c** : arrêt d'une commande

**Ctrl+z** : suspendre(mettre en pause) une commande

**CTRL+t** : corréction d'une erreur de frappe en inversant 2 lettres

**Ctrl+a** : aller au début de ligne

**Ctrl+e** : aller à la fin de ligne

**Ctrl+s** : interruption de la sortie de terminal (masquer la saisie)

**Ctrl+q** : annuler l'interruption de la sortie (afficher la saisie)

**Ctrl+u** : efface tout à gauche du curseur

**Ctrl+w** : efface le mot à gauche du curseur

**Ctrl+k** : efface le mot à droite du curseur

**Ctrl+y** : coller la saisie précédente

**Ctrl+d** : efface le caractère courant, si la ligne est vide **deconnexion**

**Alt+b** : se déplacer en avant, mot par mot dans la ligne de commande

**Alt+f** : se déplacer en arrière mot par mot dans la ligne de commande

**Alt+d** : efface le mot suivant

**Alt+t** : échange le mot courant avec le mot précédent

**Alt+c** : met en majuscule la lettre courante, tout le reste dut mot courant en minuscules, puis se deplace au mot suivant

**Alt+l** : met en majuscules à partir de la lettre courante jusqu'à la fin de mot, puis se deplace au mot suivant

**Alt+u** : met en minuscules à partir de la lettre courante jusqu'à la fin de mot, puis se deplace au mot suivant

**Alt+Backspace** : effacer le mot précédent (équivalent **Ctrl+w**)

# Les redirections et les pipelines

D'abord on va commencer avec une petite explication concernant les descripteurs des "entrées - sorties" :

* tout ce que vous écrivez dans le shell s'appelle STDIN (STandarDINput)
* tout ce que vous voyez à l'écran peut être :
	+ STDOUT (STandarDOUTput)
	+ STDERR (STandarDERRor)

Ces descripteurs sont numérotés comme suit :

 0: entrée standard (STDIN) <---------------- clavier

Processus 1: sortie standard (STDOUT) ---------------> écran

 2: sortie erreurs (STDERR) ----------------> écran

# Les redirections

Ce quoi une redirection ?
C'est la possibilité de diriger le résultat d'une commande en utilisant d'autres destinations que les descripteurs standards.

Pour réaliser une redirection on utilise :

commande > fichier - redirection en mode écriture vers le fichier
le fichier est créé s'il n'existe pas
son contenu sera remplacé par le nouveau si le fichier existe déjà

commande >> fichier - redirection en mode ajout vers le fichier
le fichier est créé s'il n'existe pas
le résultat sera ajouté à la fin de fichier

commande < fichier - la commande lit depuis le fichier .

# Les pipelines

commande1 | commande2 - le résultat de la commande1 est utilisé par la commande2

commande1 & commande2 - les commandes sont exécutées simultanément, commande1 & s'exécutant en arrière-plan

commande1 && commande2 - si la commande1 réussi la commande2 est executée

commande1 || commande2 - la commande2 s'exécute seulement si la commande1 échoue

commande1;commande2 - les commandes sont exécutées dans l'ordre

# Les scripts

Les shells ne sont pas seulement des interpréteurs de commandes mais également de véritables langages de programmation.

Un script correspond à une suite de commandes écrite dans un fichier. Ceci permet d’automatiser certaines tâches répétitives ou de faire de petits programmes adaptés aux besoins de l’utilisateur.

Ici, nous créerons un script pour le shell bash. 3 étapes suffisent :

* Créer un fichier contenant comme première ligne la syntaxe suivante:

#! /bin/bash

* Rendre ce fichier executable en utilisant la commande:

$ chmod u+x script

* Executer le script en utilisant la commande:

$ script (ou le path s'il ne se trouve pas dans $PATH) ou

$ .\script

#  Tableaux et paramètres :

Voici la syntaxe pour initialiser et pour utiliser un tableau en Bash :

mon\_tableau[index]=variable # instanciation

${mon\_tableau[index]} # appel d'un élément

${mon\_tableau[\*]} # appel de l'ensemble du tableau

Exemple :

#!/bin/bash

nom[0]='Bonjour'

nom[1]='Monsieur'

#affiche 'Bonjour'

echo ${nom[0]}

#affiche 'Bonjour Monsieur'

echo ${nom[\*]}

Les scripts Bash sont dotés de certains paramètres spéciaux. On peut récupérer leurs valeurs dans le script par :

* $0 correspond au nom du script lancé, $1 correspond au premier argument, $2 au deuxième argument ...
* $# a pour valeur le nombre total de paramètres ($0 compris) passés au script
* $? a pour valeur le code de retour de la dernière commande exécutée dans le shell
* $@ pour récupérer la concaténation de tous les paramètres, en les séparant par un espace

3. Les chaînes de caractères :

* ' ' délimitent une chaîne de caractères. A l'intérieur, tous les métacaractères perdent leur signification.
* " " délimitent une chaîne de caractères. A l'intérieur, tous les métacaractères perdent leur signification, à l'exception des métacaractères `, $ et \
* \ protège le caractère qui suit, que ce soit un caractère normal ou un métacaractère du shell (sauf à l'intérieur d'une chaîne délimitée par des ').

Exemple :

#!/bin/bash

nom='Dupont'

echo 'Mr Dupont'

echo "Mr $nom"

echo "Mr \"$nom\""

Pour connaître la taille(longueur) d'une chaîne de caractères, utilisez la syntaxe suivante :

${#ma\_variable}

Exemple :

nom='Dupont'

#affiche 6

echo ${#nom}

# Les conditions :

Cela peut paraître surprenant, mais il n'existe pas de type Booléen( True/False ou encore 0/1 ) en Bash, ni d'autres types d'ailleurs, car Bash est un langage non-typé. Ceci étant, les conditions, sous leur forme booléenne, n'éxistent donc pas à proprement parler. Dans ce document, nous avons tout de même utilisé le mot 'condition' pour ne pas perturber le lecteur, mais sachez que ce sont en fait des 'commandes' qui jouent ce rôle. Je m'explique, Bash considère qu'une commande s'est bien déroulée lorsqu'elle reçoit comme valeur de sortie '0' (zéro), toute autre valeur correspondant à une exécution non réussie (entièrement ou partiellement). Donc, si, dans la structure de commande, la commande qui joue le rôle de condition renvoie '0', alors cela correspond à un True dans le type Booléen, et vice-versa. Toute condition est analysée par la commande test(interne à Bash), sans pour autant qu'elle soit obligatoirement écrite. Voici trois exemples pour expliciter ce qui vient d'être dit : Exemples :

if test -f mon\_fichier; then

echo "le fichier existe"

fi

if [ -f mon\_fichier ]; then

echo "le fichier existe"

fi

if [ ./ma\_commande ]; then

echo "la commande s'est terminée avec succès"

fi

Quelques comparaisons utiles :

-f teste l'existence d'un fichier

-d teste l'existence d'un répertoire

-x teste si le fichier existe et est exécutable

-r teste si le fichier existe et est ouvert en lecture

-w teste si le fichier existe et ouvert en écriture

-s teste si le fichier existe et a une taille supérieur à 0 octet

Pour les comparaisons arithmétiques (voir la deuxième partie de ce tutoriel pour plus d'informations), il peut être plus agréable d'utiliser une autre syntaxe :

if (( $nombre\_1 == $nombre\_2 )); then

echo "il y a egalité"

fi

Remarque : il faut TOUJOURS laisser un espace entre les crochets "[ ]" et entre chaque argument; sinon vous obtiendrez des méssages d'erreurs tels que "syntax error in expression" ou "syntax error near unexpected token".

#  Les structures de contrôles :

Comme dans tout langage de programmation, avec Bash il est possible de casser la séquence d'exécution des commandes, en utilisant les structures de contrôles habituelles. Voici la syntaxe à utiliser.

**IF :**

if condition\_1; then

commandes1

elif condition\_2; then

commandes2

else

commandes3

fi

**FOR :**

for variable in liste\_de\_cas do

commandes

done

for (( expr1 ; expr2 ; expr3 )) do

commandes

done

 **WHILE**

while condition do

commandes

done

**UNTIL :**

until condition do

commandes

done

**CASE**

case variable in

cas1)

commande1

;;

cas2)

commande2

;;

\*)

commande\_par\_defaut

;;

esac

( cas1 et cas2 sont des chaînes de caractères. )

 **Les fonctions :**

On peut définir des fonctions en Bash, cela peut être très utile pour structurer ses programmes. La syntaxe est la suivante :

ma\_fonction() {

corps

}

Pour appeler la fonction dans le script ce sera :

ma\_fonction param1 param2 param3 ...

A l'intérieur du corps de la fonction, les paramètres sont disponibles dans les variables $0, $1, $2 ... Le nombre de paramètres etant toujours $#. Une variable utilisée à l'intérieur d'une fonction est globale! Pour éviter ce phénomène, il faut rajouter local à la déclaration de la variable :

local MA\_VARIABLE

Exemple :

#!/bin/bash

valeur=0

calcul() {

somme=$(( $1 + $2 ))

local valeur=$somme

}

calcul 5 9

#retourne "14"

echo $somme

#retourne "0"

echo $valeur

Une fonction se termine soit après l'exécution de la dernière commande située avant l'accolade fermante, auquel cas le code de retour est celui de cette dernière commande, soit après exécution d'une commande return n, avec "n" un nombre compris entre 1 et 255. Remarque : la déclaration de la fonction dans le script doit ABSOLUMENT se faire avant son appel, autrement, vous obtiendrez un "command not found". Pour utiliser vos fonctions depuis d'autres scripts: Exemple fichier mesfonctions.inc.sh

ma\_fonction() {

mavariable1=$1

mavariable2=$2

}

Exemple d'inclusion dans un autre script sh: Remarque: il faut que le sh inclus soit dans le même répertoire que le sh appelant.

# On inclut le fichier de fonctions

. ./mesfonctions.inc.sh

# On définit les paramètres à passer à la fonction

mon\_parametre1="mon premier parametre"

mon\_parametre2="mon deuxieme parametre"

# On appelle la fonction avec ces parametres

ma\_fonction mon\_parametre1 mon\_parametre2

#suite du code...