

Unix/GNU Linux

Notions de base

Thierry Vaira

BTS SN La Salle Avignon

v1.2 - 13 novembre 2018



UNIX

- Le premier système UNIX date de 1969 (Laboratoires Bell d'AT&T, Ken Thompson & Dennis Ritchie)
- Système d'exploitation multitâches et multi-utilisateurs
- Deux interfaces utilisateurs :
 - GUI (*Graphical User Interface*)
 - CLI (*Command Line Interface*)

Le mode console est l'interface utilisateur de base du système d'exploitation Unix (en plein écran). Le mode terminal émule une console qui s'affiche en général dans une fenêtre de l'écran.

Famille UNIX

UNIX désigne une super famille de systèmes qui se compose de trois grandes familles :

- Système V (AT&T)
- BSD (*Berkeley Software Distribution*)
- Les clones : Minix, **GNU/Linux**, XINU,

On nomme « famille Unix », « systèmes de type Unix » ou simplement « systèmes Unix » l'ensemble de ces systèmes. Il existe un ensemble de standards réunis sous les normes **POSIX** et *single UNIX specification* qui visent à unifier certains aspects de leur fonctionnement. Le nom « UNIX » est maintenant une marque déposée de l'*Open Group*.

➡ <http://www.levenez.com/unix/>

Composants principaux

Le système d'exploitation **GNU/Linux** est composé :

- ⇒ d'un noyau (*kernel*)
- ⇒ d'un *shell* et des applications (*user*)
- ⇒ d'un système de fichiers
- ⇒ de la mémoire virtuelle, composée de la RAM physique et de la zone de *swap*, généralement stockée sur un disque dur local.

Les tâches (services) du système d'exploitation sont assurées par des processus qui fonctionnent en permanence (tâche de fond) dans le système : les **démons** (*daemons*).

Le noyau

Le noyau est en charge des opérations de base suivantes :

- gestion des périphériques (à travers les pilotes de périphériques, *driver*), de la mémoire, des processus et démons,
- contrôle des échanges (des flux de données, par exemple TCP/IP) entre les utilitaires du système et le matériel,
- séquençage et exécution des processus et partage du temps des processeurs (*scheduler*),
- gestion de la mémoire virtuelle.

Le shell et les applications

Le *shell* constitue l'interface entre le noyau et l'utilisateur. Les *shells* disponibles (les plus utilisés) sous Ubuntu Linux sont :

- Le *Bourne Shell* (**sh**) : développé pour l'environnement UNIX®, il constitue le *shell* le plus utilisé pour l'écriture des scripts.
- Le *C Shell* (**cs****h**) : version améliorée du *Bourne Shell* avec une syntaxe proche du langage C.
- Le *GNU Bourne Again Shell* (**ba****sh**) : *shell* compatible avec le *Bourne Shell* qui incorpore les fonctionnalités telles que l'historique des commandes, les alias et l'édition de la ligne de commande. C'est le *shell* **par défaut** de Ubuntu Linux.

Le système de fichiers

Le système de fichiers sous Linux se compose d'une hiérarchie de répertoires, sous-répertoires et fichiers. Le répertoire le plus élevé dans l'arborescence est nommé la racine (*root*) symbolisé par `/`. L'arborescence est unique. Quelques répertoires importants :

- `/etc` : contient les fichiers de configuration du système et des applications
- `/dev` : contient les fichiers spéciaux de périphériques qui représente les points d'accès au matériel
- `/bin` : contient les commandes de base du système
- `/sbin` : contient les outils systèmes pour l'administration
- `/usr` : contient les commandes et applications pour les utilisateurs, dont les environnements graphiques
- `/home` : contient les répertoires personnels des utilisateurs
- `/var` : contient les fichiers dont le contenu varie en fonction de l'utilisation du système (bases de données, fichiers de logs, ...)
- `/proc` : représente le point d'accès aux informations (variables, tables, liste ...) du noyau et des processus



Zone de swap

La zone de *swap* est utilisée lorsque la mémoire physique (RAM) est remplie. Si le système a besoin de plus de ressources mémoires et que la mémoire physique est remplie, les pages de mémoire (page = bloc de taille fixe) inactives (non utilisées depuis un certain temps) sont déplacées dans la zone de *swap*.

Quelques règles de base pour la zone de *swap* :

- Qu'elle soit stockée dans une partition dédiée,
- La partition dédiée à la zone de *swap* soit sur un disque (voire un contrôleur) différent de la partition système et des données les plus utilisées,
- Que sa taille soit au minimum la taille de la mémoire physique.



Système de fichiers

- Un système de fichiers (*file system*) définit l'organisation d'un disque (ou plus précisément d'une partition d'un disque).
- C'est une structure de données permettant de stocker les informations et de les organiser dans des fichiers sur ce que l'on appelle des mémoires secondaires (disque dur, disquette, CDROM, clé USB, disques SSD, etc.).
- Il existe de nombreux systèmes de fichiers différents : ext2, ext3, ext4, UFS, HFS, reiserfs, etc.
- Chaque fichier est décrit par des métadonnées (conservées dans l'**inode** sous Unix), alors que le contenu du fichier est écrit dans un ou plusieurs blocs (taille fixe) du support de stockage selon la taille du fichier. Le terme inode désigne le descripteur d'un fichier sous UNIX.
- La commande `stat` permet d'afficher l'intégralité du contenu de l'inode.



Type de fichiers

- Sous Unix, un fichier n'est pas structuré : c'est une suite d'octets. On distingue en général deux types de fichiers : **texte** et **binaire**.
- Les **fichiers textes** ont un contenu pouvant être interprété comme du texte (caractères la plupart du temps codés en ASCII). On utilise habituellement un éditeur de texte pour manipuler ce type de fichiers. Exemples de fichiers textes : code source d'un programme, scripts, fichiers de configuration, etc .
- Les **fichiers binaires** sont tous les fichiers autres que des fichiers textes. Le contenu d'un fichier binaire correspond souvent à un format précis lié à l'usage d'un logiciel applicatif spécifique. Exemples de formats binaires usuels : fichiers exécutable (code machine), fichiers de base de données, fichiers multimédias : images, sons, vidéos, documents textes (traitement de texte), etc.



Autres fichiers

- Un fichier (texte ou binaire) qui a subi une transformation par un algorithme en vue de diminuer sa taille est appelé **fichier compressé**. Le fichier transformé est un fichier binaire.
- Une **archive** est un fichier dans lequel se trouve regroupé des fichiers ou tout le contenu d'une arborescence. Le but principal d'une archive est de tout contenir (données + descriptions) en un seul fichier. Le fichier archive est un fichier binaire. Les archives sont souvent compressées.

Type de fichiers

Sous UNIX, TOUT EST FICHIER !

Mais, le système d'exploitation distingue les types de fichiers suivants :

- les fichiers « normaux » (*regular*) : texte ou binaire (-)
- les fichiers « répertoires » (*directory*) (d)
- les fichiers « périphériques » (*device*) (c ou (b))
- les fichiers « liens symboliques » (l)
- les fichiers « *socket* » (s)
- les fichiers « tubes nommés » (*pipe*) (p)

Le type de fichier est indiqué dans l'inode (commande `stat`) ou visualisé par la commande `ls -l`.

Exemple

```
$ ls -il Makefile
13107382 -rw-r--r-- 1 tvaira tvaira 22074 mai 22 18:06 Makefile

$ stat Makefile
Fichier : « Makefile »
  Taille : 22074      Blocs : 48      Blocs d'E/S : 4096 fichier
Périphérique : 805h/2053d Inoeud : 13107382 Liens : 1
Accès : (0644/-rw-r--r--) UID : ( 1029/ tvaira) GID : ( 1000/ tvaira)
  Accès : 2017-05-09 08:18:56.725487886 +0200
Modif. : 2017-05-22 18:06:04.921242241 +0200
Changt : 2017-05-22 18:06:04.921242241 +0200
  Créé : -

// Attention :
$ stat Makefile --printf="Nb blox = %b (avec taille bloc : %B octets)\n"
Nb blox = 48 (avec taille bloc : 512 octets)
```

Remarque : les inodes des fichiers ne contiennent pas les noms des fichiers



Les utilisateurs

- Sur UNIX, chaque utilisateur est identifié par un nom, par un UID (*User IDentification*) et par un GID (*Group IDentification*).
- Il peut appartenir à plusieurs groupes, eux-mêmes identifiés par un nom et par un GID.
- Comptes utilisateurs locaux définis dans les fichiers : `/etc/passwd`, `/etc/shadow` et `/etc/group`
- Commandes : `id`, `whoami`, `who`, `who am i`, `w`, `last`, `users`, ...

Les processus

- Un processus est l'image (code + données + informations PCB) en cours d'exécution d'un programme binaire.
- Chaque processus en cours d'exécution est identifié par un PID (*Process IDentification*) et un PPID (*Parent Process IDentification*).
- Les processus sont donc organisés de façon hiérarchisée chacun d'entre eux à un seul et unique parent. Il existe donc un processus père de tous les autres ayant comme PID 1 (historiquement le programme `init`).
- Dans un système multitâches, les processus sont gérés par un *scheduler* (ordonnanceur).
- Commandes : `ps`, `top`, `jobs`, `pidof`, `kill`, `killall`, `pkill`, ...



Les entrées/sorties

- Lorsqu'un processus est créé, trois fichiers (flux) sont ouverts et associés au processus automatiquement :
 - En lecture, l'entrée standard (*stdin*) (descripteur 0) : par défaut le clavier
 - En écriture, la sortie standard (*stdout*) (descripteur 1) : par défaut l'écran
 - En écriture, la sortie d'erreurs (*stderr*) (descripteur 2) : par défaut l'écran
- Les entrées/sorties peuvent être redirigées (par le *shell* avec *>*, *»*, *<*, ...)
- Les processus peuvent donc communiquer entre eux par l'intermédiaire de ces entrées/sorties standards : tube (*pipe*) |



- « *Il est plus facile de définir un système d'exploitation par ce qu'il fait que par ce qu'il est.* » J.L. Peterson.
- « *Unix est convivial. Cependant Unix ne précise pas vraiment avec qui.* » Steven King
- « *Unix n'a pas été conçu pour empêcher ses utilisateurs de commettre des actes stupides, car cela les empêcherait aussi des actes ingénieux.* » Doug Gwyn
- « *Unix ne dit jamais 's'il vous plaît'.* » Rob Pike
- « *Unix est simple. Il faut juste être un génie pour comprendre sa simplicité.* » Denis Ritchie

Philosophie UNIX ou « *less is more* »

- Des programmes qui effectuent une seule chose et qui le font bien
- Le silence est d'or
- Des programmes qui collaborent
- Des programmes pour gérer des flux de texte
- « Qu'est-ce qu'un UNIX ? », je répondrais par ce type de commande (pleine de magie et d'intelligence) : `history | grep -v " h" | sed 's/[\t]*$//' | sort -k 2 -r | uniq -f 1 | sort -n`

[Extrait d'un article de Denis Bodor dans GNU/Linux Magazine HS n°46]

Les paquetages (*packages*) DEBIAN

- Tous les logiciels qui composent Ubuntu Linux sont fournis sous forme de paquetages (*packages*) (.deb).
- Un paquet (*package*) contient :
 - des fichiers décrivant le *package* (description, version, signature, dépendances, ...)
 - les fichiers à installer
 - des scripts qui s'exécutent avant ou après l'installation ou la suppression



Voir aussi : RPM (*Red Hat Package Manager*), .tgz (*Slackware*), Pacman (*PACkage-MANager*), ...



Les gestionnaires de paquets DEBIAN

- `dpkg` est un outil pour l'installation, la création, la suppression et la gestion des paquets Debian.
- `aptitude` est la principale interface CLI à `dpkg`.
- APT est un système de gestion de paquets logiciels.
- `synaptic` est la principale interface GUI à APT.

Utilisation

- Ouvrir une session locale : `login`, `su`, ...
- Ouvrir une session distante : `telnet`, `ssh`, ...
- L'invite ou *prompt* (PS1) : `#` ou `$`
- Les commandes (PATH)
- Les interpréteurs de commandes (SHELL)
- Fermer une session : `logout`, `exit`,  + 

CONSULTEZ LE MANUEL (RTFM)

- La commande `man` : `man man`, `man ls`, ...
- La commande `apropos` : recherche dans la totalité du manuel
- La commande `whatis` : affiche la description des pages de manuel
- La commande `help` : affiche de courts résumés des commandes intégrées (*shell*).
- L'option `--help` : affiche l'aide-mémoire d'une commande
- La commande `info` : `info info`, `info ls`, ...

Assiste moi

La touche  assure la complétion en mode CLI :

- des commandes
- des options
- des paramètres : noms de fichier, chemin (PATH)
- des variables d'environnement

Manipuler les répertoires

- `pwd` : le chemin (*path*) complet
- `mkdir`, `rmdir` : créer, supprimer
- `cd` : se déplacer
- `mv` : déplacer, renommer
- `chmod` : modifier les droits et permissions d'accès
- `chown`, `chgrp` : modifier le propriétaire

Notions de chemin

Chemins relatifs :

- . : répertoire courant ("ici")
- .. : répertoire parent ("au dessus")
- ~ ou \$HOME : répertoire personnel ("chez moi")

Chemin absolu :

- / : la racine (root)

Manipuler les fichiers

- `ls` : lister le contenu d'un répertoire
- `cat`, `more`, `strings`, ... : afficher le contenu texte d'un fichier
- `touch` : créer un fichier vide (modifier l'horodatage d'un fichier)
- `mv`, `rm` : déplacer, renommer, supprimer
- `find` : rechercher des fichiers
- `ed`, `vi`, `vim` : éditer un fichier
- `chmod` : modifier les droits et permissions d'accès
- `chown`, `chgrp` : modifier le propriétaire

Contrôler l'accès aux fichiers

Sous UNIX, il existe deux types de sécurité pour les fichiers et répertoires : les droits et permissions UNIX et les ACL (*Access Control List*).

Il y a trois types de permissions :

- r : accès en lecture
- w : accès en écriture
- x : exécution (fichier), traversée (répertoire)

Chacune de ces permissions peuvent être attribuée à :

- u : l'utilisateur (propriétaire)
- g : le groupe (propriétaire)
- o : les autres
- a : tout le monde

Manipuler les fichiers

Droits et permissions supplémentaires UNIX :

- SETUID : exécution avec l'UID du propriétaire du fichier, pas d'effet sur les répertoires
- SETGID : exécution avec l'GID propriétaire du fichier, création d'un répertoire avec le GID propriétaire du répertoire parent
- BIT STICKY : suppression restreinte pour le répertoire, conserver l'image d'un programme en mémoire

Les commandes :

- `chmod` : modifier les droits et permissions UNIX
- `umask` : fixer le masque de création de fichiers

Gérer les processus

Une commande, une fois lancée, devient un processus (**programme en cours d'exécution**)

- `ps`, `top`, `jobs` : lister les processus
- `pidof` : afficher le PID d'un processus
- `kill`, `killall`, `pkill` : envoyer un signal (`kill -1`) à un processus ou plusieurs (dont : interrompre, stopper, terminer ou tuer)
- `&` à la fin de la ligne de commande : lancer une commande en arrière plan
- `nohup` : détacher le processus de la console
- `Ctrl` + `Z` , `fg`, `bg` : mettre en pause, déplacer une tâche au premier plan, en arrière plan
- `at` : lancer des commandes à une heure précise (différé)
- `batch` : exécuter des commandes lorsque la charge système le permet
- `cron` : planifier l'exécution de commandes (périodiques)

Les expressions rationnelles

- Les expressions rationnelles (*Regular Expressions*) sont beaucoup utilisées sous UNIX, et notamment avec les outils d'éditions de texte et les filtres (sed, grep, awk, ...).
- Il s'agit d'un mécanisme qui permet de décrire des ensembles de caractères dans le cadre d'une recherche ou d'un remplacement de texte.
- Une expression rationnelle est une suite de caractères qu'on appelle plus simplement motif (*pattern*) pour trouver une correspondance (*match*).
- Les mécanismes de base pour former un motif sont basés sur des caractères spéciaux de substitution, de groupement et de quantification.

Les quantificateurs I

- ? qui définit un groupe qui existe zéro ou une fois : toto?
correspondant (*match*) alors à « tot » ou « toto » mais pas « totoo » ;
- * qui définit un groupe qui existe zéro ou plusieurs fois : toto*
correspondant à « tot », « toto », « totoo », « totooo », etc. ;
- + qui définit un groupe qui existe une ou plusieurs fois : toto+
correspondant à « toto », « totoo », « totooo », etc. mais pas « tot ».
- {n} qui définit exactement *n* occurrences de l'expression précédant les accolades : a{3} correspondant à « aaa » mais pas à « aa », ni « aaaa » etc . ;

Les quantificateurs II

- $\{n,m\}$ qui définit entre n et m occurrences de l'expression précédant les accolades : $a\{2,4\}$ correspondant à « aa », « aaa », « aaaa » mais pas à « a », ni « aaaaa » ;
- $\{n,\}$ qui définit au moins n occurrences de l'expression précédant les accolades : $a\{3,\}$ correspondant à « aaa », « aaaa », « aaaaa » mais pas à « aa » ;

Remarque : Pour neutraliser un caractère spécial, il faut l'« échapper », c'est-à-dire le faire précéder du caractère \ (anti-slash).

Les opérateurs de base |

- l'opérateur de concaténation de deux expressions (implicite) : `ab` correspondant à « `ab` » mais pas à « `a` », ni « `b` », ni une chaîne vide ;
- `.` qui définit un caractère et un seul : `.` correspondant à « `a` », « `b` », ... mais pas une chaîne vide, ni « `ab` » ;
- `|` qui est l'opérateur de choix entre plusieurs alternatives. Il peut être combiné autant de fois que nécessaire pour chacune des alternatives possibles. Il fait correspondre l'une des expressions placées avant ou après l'opérateur : `a|b` correspondant à « `a` », « `b` » mais pas à une chaîne vide, ni « `ab` », ni « `c` » ;

Les opérateurs de base II

- `[]` qui définit un des caractères entre crochets (« classe de caractères ») : `[aeiou]` correspondant à « a », « e », « i », ... mais pas à une chaîne vide, ni « b », ni « ae » ; Entre ces crochets, un intervalle de caractères peut être indiqué en donnant le premier et le dernier caractère, séparés par un tiret : `[a-d]` est équivalent à `[abcd]`
- `[^]` qui définit un caractère n'étant pas entre crochets (négation) : `[^aeiou]` correspondant à « b », ... mais pas à une chaîne vide, ni « a », ni « e », ... ni « bc » ;
- `()` qui définit un groupement de l'expression entre parenthèses : `(détecté)` correspondant à « détecté » mais pas à « détect », « détecta », « détectés » ;

Les opérateurs de base III

- `^` qui ne correspond à aucun caractère mais fixe une condition en indiquant que ce doit être au début d'une ligne : `^a` trouve « a » en début de ligne mais pas dans « ba » ;
- `$` qui ne correspond à aucun caractère mais fixe une condition en indiquant que ce doit être à la fin d'une ligne : `a$` trouve « a » en fin de ligne mais pas dans « ab » ;
- Entre les crochets `[]`, les métacaractères sont interprétés de manière littérale : `[.?*]` désigne l'ensemble constitué des caractères « . », « ? » et « * ».
- Les sous-ensembles placés entre `(` et `)` peuvent être rappelés par un numéro correspondant à leur ordre de déclaration, précédé du caractère `\`.

Les standards I

Le standard POSIX propose plusieurs normes :

- BRE (*Basic Regular Expressions*) pour les expressions rationnelles basiques. C'est par exemple le standard par défaut pour sed et grep.
- ERE (*Extended Regular Expressions*) pour les expressions rationnelles étendues. C'est l'option -E pour grep et -r pour sed.

Les expressions rationnelles de **Perl** sont également un standard de fait, en raison de leur richesse et de leur puissance (elles ont donné la bibliothèque **PCRE**). C'est l'option -P pour grep.

Les notations peuvent varier légèrement d'un moteur d'expressions rationnelles à l'autre. Un moteur d'expressions rationnelles est un outil permettant de manipuler des expressions rationnelles.

Les standards II

- Dans BRE, les accolades, les parenthèses, le symbole « ? » et le symbole « + » ne sont pas des métacaractères : ils ne représentent qu'eux même. Pour prendre leur notion de métacaractères, ils ont besoin d'être échappés par le symbole « \ ».
- Contrairement aux expressions rationnelles basiques, la norme ERE reconnaît les caractères vus précédemment comme des métacaractères. Ils doivent ainsi être échappés pour être interprétés littéralement.
- La plupart des exemples donnés ici étaient des expressions régulières étendues POSIX.

Les classes de caractères I

Les classes de caractères les plus utilisées sont généralement fournies avec le moteur d'expression régulière.

Quelques classes de caractères POSIX prédéfinies :

- `[:digit:]` = chiffres décimaux
- `[:alpha:]` = caractères alphabétiques
- `[:alnum:]` = caractères alphanumériques
- `[:blank:]` = espace et tabulation
- `[:space:]` = caractères d'espacement
- `[:punct:]` = caractères de ponctuation
- etc ...

⇒ `[:alnum:]` est équivalent à `[0-9A-Za-z]` pour des caractères ASCII.

Les commandes

- Grep (*Global Regular Expression Printer*) permet de faire des recherches de lignes contenant une chaîne correspondant à une expression rationnelle.
- Sed (*Stream Editor*) est un éditeur qui possède les mêmes fonctionnalités que l'éditeur ed mais qui ne travaille pas en mode interactif. Il permet donc, contrairement à grep, de modifier le flux de lignes qui lui est passé.
- etc ...

Exemple

Utilisation d'expressions rationnelles avec grep et sed sur un fichier texte contenant des codes postaux :

```
$ cat liste.txt
Sarrians 84260
Avignon 84000
Carpentras 84200
Jonquières 84150
Marseille 13000
Istres 13800
Vitrolles 13127
Paris 75000
```

```
// sed en mode ERE (extended)
$ cat liste.txt | sed -rn '/(84|13)[[:digit:]]{3}/p'
// grep en mode BRE (basic)
$ cat liste.txt | grep '\(84\|13\) [[:digit:]]\{3\}'
// grep en mode ERE (extended)
$ cat liste.txt | grep -E '(84|13)[[:digit:]]{3}'
Sarrians 84260
Avignon 84000
Carpentras 84200
Jonquières 84150
Marseille 13000
Istres 13800
Vitrolles 13127
```


Shell script

- Les scripts sont des fichiers textes contenant des suites de commandes exécutées par un *shell*.
- Ils sont surtout utilisés par les administrateurs réseaux et les développeurs car ils permettent d'automatiser des traitements.
- Il existe plusieurs manières d'exécuter un script :
 - le rendre exécutable :
`$ chmod +x monscript ; ./monscript`
 - passer son nom en paramètre d'un *shell* :
`$ sh monscript`
 - utiliser une fonction de lancement de commande du *shell* :
`$. monscript` ou `$ source monscript`

Syntaxe

Dans un script :

- un commentaire commence par le caractère # et finit à la fin de la ligne
- une instruction nulle est indiquée par le caractère :
- le chemin du *shell* avec lequel il doit être exécuté précédé des caractères #! (le *shebang*) est indiqué sur la première ligne
(si ce n'est pas le cas, c'est le *shell* à partir duquel le script est lancé qui s'en charge)

Arguments

Les arguments sont les paramètres tapés après le nom du script exécuté. Ils sont accessibles et manipulables au travers de variables prédéfinies :

- \$# nombre d'arguments
- \$* liste de tous les arguments
- \$0 nom du script en cours d'exécution
- \$1 premier argument, \$2 deuxième argument, \$3 troisième argument, ...
- @\$: liste de tous les arguments à partir de \$1

Structures de contrôle

- Les structures conditionnelles : `if then else fi`, les choix multiples avec `case` ou `select`
- Les contrôles itératifs : les boucles `for`, `while` et `until`
- Les structures de contrôles utilisent la commande externe `test` pour évaluer des expressions booléennes. Elle renvoie '0' si le test est vrai, et '1' sinon. La forme la plus courante est l'utilisation de crochets `[` (ou `[[` depuis la version 2) qui encadrent le test (délimités par au moins un espace).

```
$ help test
```

```
$ help [[
```

✍ Avec `[[`, l'opérateur `=~` permet de tester une expression rationnelle.



Commandes internes

Un certain nombre de commandes sont exécutées directement par le *shell* et ne sont pas des programmes externes.

Les plus utiles :

- `echo` : affiche sur la sortie standard (voir aussi `printf`)
- `read` : lit l'entrée standard (et stocke dans des variables les mots tapés au clavier par l'utilisateur)
- `exit` : termine le script immédiatement en retournant une valeur (0 par défaut)
- `eval` : exécute des arguments comme s'ils étaient une commande. Fonctionne comme les guillemets inversés “.
- `let` : sert à évaluer des expressions arithmétiques.

Commandes externes

- `test` : permet d'évaluer des expressions booléennes. Il y a trois types de tests : sur les fichiers, sur les chaînes de caractères et sur les nombres.
- `expr` : sert à évaluer des expressions, et notamment les expressions arithmétiques. Une autre manière d'évaluer une expression arithmétique dans un script est d'utiliser la notation suivante :
`$((expressions))`
- `bc` : calculatrice pour notamment pour des calculs complexes ou sur des réels (avec un *pipe* et les options `bc -lq`)

Les variables

- Une variable existe dès qu'on lui attribue une valeur.
- Une chaîne vide est une valeur valide.
- Une fois qu'une variable existe, elle ne peut être détruite qu'en utilisant la commande interne `unset`.
- Une variable peut recevoir une valeur par une affectation de la forme :
`nom=[valeur]`
- Le caractère '\$' permet d'introduire le remplacement des variables :
`echo $nom`
- La substitution de commandes permet de remplacer le nom d'une commande par son résultat. Il en existe deux formes : `$(commande)` ou `'commande'`