Linux pour la programmation

Christophe Labourdette
christophe.labourdette(at)cmla.ens-cachan.fr

Septembre 2016

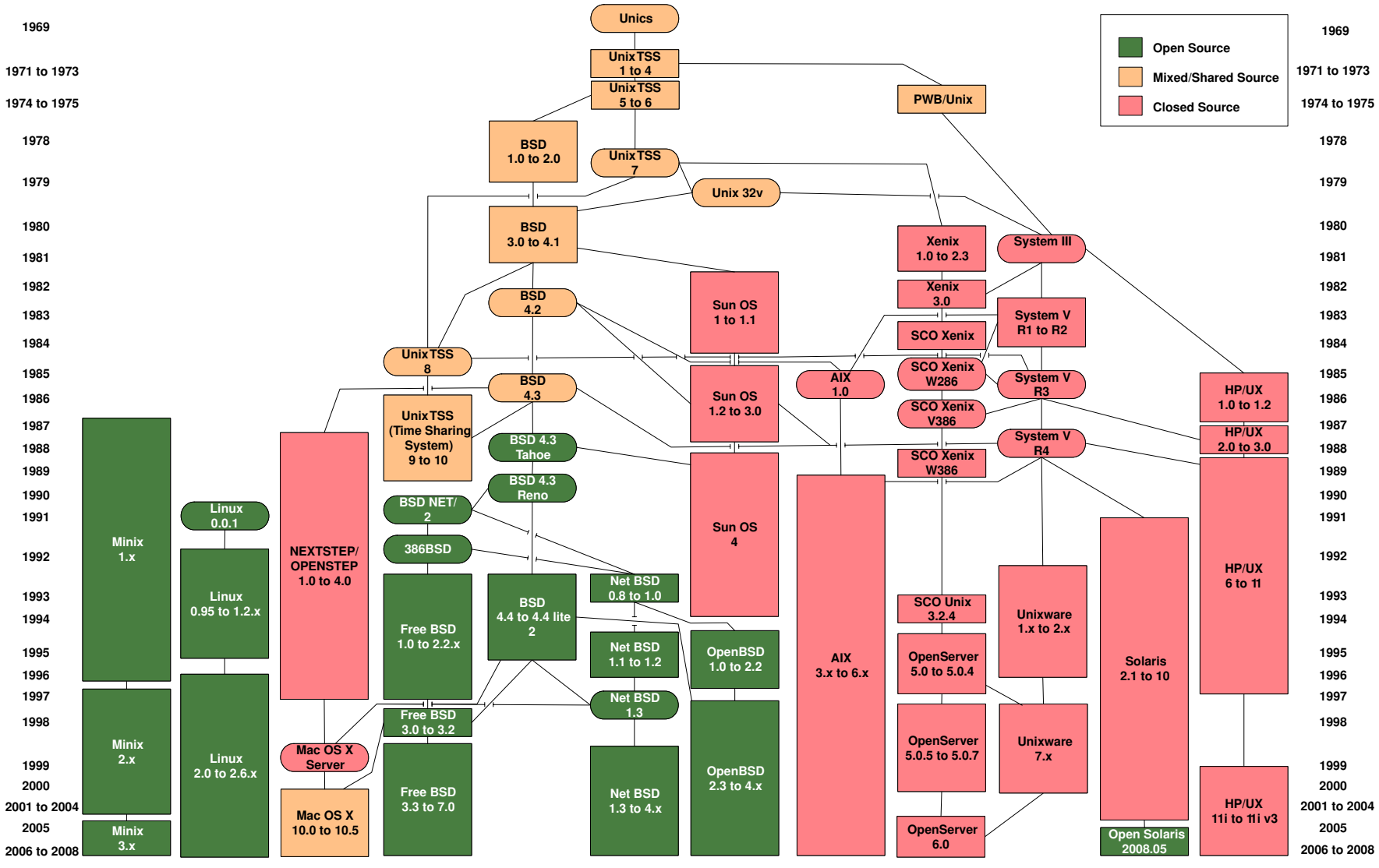
Système

Le système d'exploitation est l'application logicielle qui gère l'ordinateur.
Il met à la disposition de l'utilisateur les ressources de la machine, matérielles ou logicielles.
Bref, on ne peut rien faire sans lui ...
Unix est un système d'exploitation, Windows aussi.

Unix

- Le système d'exploitation Unix est né en 1969 au Bell Labs.
- Nombreuses versions différentes dans les années 70 et 80.
- Linux est conçu comme une version libre et gratuite d'UNIX.
- LINUX est né en 1991 à l'Université d'Helsinki en Finlande.
- Linux est un système Unix
- Il existe des systèmes Unix libres autres que Linux (OpenBSD, FreeBsd, ...)

La Famille Unix



Linux

- Linux est un noyau,
- Linux est un système multi-tâches,
- Linux est un système multi-utilisateurs,
- Linux est un système opensource,
- Linux est un système UNIX,
- Linux est un système évoluant trés rapidement,
- Linux est un système conçu pour Internet.

Distributions

Les systèmes Linux sont disponibles sous forme de distribution, elles n'ont pas toutes le même comportement

linux

- mandriva, fedora, aurox, centos
- debian, ubuntu, knoppix
- suse
- slackware, gentoo

bsd

On trouve aussi des distributions à base de systèmes BSD : freeBSD, OpenBSD, NetBSD

Qu'est-ce qu'une distribution

Système

Le coeur du système Linux, c'est un noyau, un système de fenêtrage, ainsi qu'un certain nombre d'utilitaires pour gérer les périphériques.

Logiciels

Bien entendu, il existe également de trés nombreux logiciels proposant des fonctionnalités dans tous les domaines, qui peuvent être ajoutés.

Assemblage

On peut comparer une distribution à un assemblage de composants disponibles autour du noyau et du système de fenêtrage.

Maintenance

Paquets

Le plus souvent, un système de paquets (qui n'est pas forcément le même selon les distributions), permet d'installer et/ou de mettre à jour les applications, qu'elles soient systèmes ou accessoires.

Mise à jour

Les mises à jour doivent être régulières, surtout celles qui concernent des failles de sécurité.

Dérive

Au fil des installations et des mises à jours le système installé devient personnalisé et spécifique, il s'éloigne de la distribution brute.

Catégories

Il est souvent possible, au sein même des distributions de choisir une catégorie. Par exemple, sur un serveur en production, il sera plus raisonnable de mettre une distribution *stable*, alors que sur un ordinateur personnel, il sera souvent préférable d'installer une version *instable*. Dans le premier cas c'est la robustesse et la qualité de service qui prime, alors que pour la version instable, l'important va plutôt être les fonctionnalités disponibles. Les mises à jours seront donc beaucoup plus fréquentes sur les versions instables et les applications pourront être en version *beta*.
Il existe dans certaines distributions d'autre catégories intermédiaires entre stable et instable.

kernel.org

Le noyau Linux fait l'objet d'un développement particulier, les contributeurs autour de Linus Torvalds possèdent leur propre site `http://www.kernel.org`
C'est eux qui décident de la sortie officielle des versions qui seront ensuite adaptées dans les distributions.
Il peut exister plusieurs branches de développement dont les rythmes ne sont pas forcément réguliers.

Xorg

Historiquement, le système de fenêtrage du système Unix s'appelle Xwindow system, actuellement l'implémentation OpenSource utilisée au sein des distributions est Xorg, elle possède son site Web http ://www.x.org
Le cycle de développement de Xorg se fait indépendamment de celui du noyau linux.
L'environnement graphique est aujourd'hui devenu partie intégrante de la distribution. Seuls les serveurs sont installés sans bureau graphique.
De même que pour le noyau Linux, la partie centrale de l'environnement graphique est entourée d'un grand nombre d'applications apportant de nouvelles fonctionnalités.

Environnement de bureau

Aujourd'hui il existe en gros deux alternatives d'environnement de bureau graphique sous Linux :

- KDE lancé en 1996 et critiqué pour son utilisation de la librairie QT qui à l'époque n'était pas d'usage libre.
- Gnome lancé en 1997, comme alternative totalement libre, en choisissant la librairie Gtk.

KDE et Gnome ont une approche différente de ce que doit être une interface graphique :

- KDE se veut complet dans l'intégration et les possibilités de configuration.
- GNOME se veut épuré et met en avant des applications et des fonctionnalités au détriment d'autres.

Souvent

La plupart des utilisateurs utilisent KDE. Il est installé par défaut dans de nombreuses distributions.
Il est tout à fait possible d'utiliser des applications développées pour l'un des environnements, en utilisant l'autre bureau.
Les applications et interfaces de configurations sont beaucoup plus nombreuses sous KDE, on peut par exemple citer :

- kdevelop (environnement de programmation),
- kile (environnement LaTeX),
- koffice (une suite bureautique),
- Amarok (Lecteur audio) ,
- Kaffeine (environnement TV et Vidéo),
- K3b (gravure CD et DVD),
- ...

Connexion

Pour utiliser une machine sous Unix/Linux, il faut se *connecter*.
Un compte comprend les données suivantes :

compte

- login : le nom de login est souvent le nom de famille
- passwd : un mot de passe
- shell : le shell de login
- home directory : un répertoire de travail dans l'arborescence
- uid : vous n'êtes qu'un numéro !
- gid : le groupe par défaut

Le shell

Le shell est l'application qui fait l'interface entre l'utilisateur et le noyau Unix. Il permet de manipuler les processus et les fichiers. Chaque connexion lance une nouvelle instance du shell de login.
Il existe plusieurs shells disponibles, ils sont tous compatibles mais diffèrent par quelques fonctionnalités. Le shell de Linux est traditionnellement *bash*.
Il est également possible de faire un peu de programmation avec le shell.

Administration

Sous Unix, l'administrateur système est l'utilisateur root qui possède l'uid 0. Il a tous les droits , sur les processus et les fichiers (;-)).
Souvent le privilège de l'administrateur est partagé et les gens passent temporairement administrateur, à l'aide d'une commande *sudo*.
En général chaque utilisateur doit signer une charte (règlement) d'utilisation qui rappelle essentiellement que :
ce n'est pas parce que vous pouvez le faire que c'est autorisé !
Tout ce qui n'est pas explicitement autorisé est interdit !

Les commandes

Il est utile de connaître quelques règles/usages à propos des commandes sous Unix et de leur paramêtrage :

- les commandes sont en minuscules.
- généralement les options sont données après la commande, à la suite d'un '-'.
- l'option -h (ou -help ou –help) indique en général la liste des options possibles.
- pour connaître la liste des commandes commençant par un ou plusieurs caractères, il suffit de taper le ou les caractères puis la touche 'TAB' deux fois.
- lorsque cela est envisageable, la commande peut fonctionner comme un filtre.

Manipuler des processus

Les processus sont identifiés par un numéro, le pid.
Pour obtenir la liste des processus on utilise *ps aux* (on remarquera qu'il s'agit d'une entorse à la règle, «les options sont précédées d'un '-'»).
On peut envoyer un signal (SIG) à un processus (PID) à l'aide de la commande *kill* : "kill -SIG PID". Voici quelques exemples :

Signal	Numéro	Description
SIGKILL	9	Tue le processus
SIGTERM	15	Termine le processus
SIGSTOP	17	Arrête momentanément
SIGCONT	19	Reprend l'exécution après un stop

TABLE : Les signaux les plus courants

entrées-sorties

Lors de l'exécution d'un processus, il y a création automatique de trois "fichiers" particuliers par le shell :

- L'entrée standard (stdin)
- La sortie standard (stdout)
- L'erreur (stderr)

Il est donc possible de rediriger ces fichiers, mais aussi de chaîner deux processus, P1 et P2, en indiquant que l'entrée standard de P2 correspond à la sortie standard de P1, c'est ce que l'on appelle un pipe. La syntaxe est la suivante P1 | P2.

Contrôle des processus

Par défaut le shell associe la sortie standard et l'entrée standard d'un processus, à la fenêtre courante. Il est possible de libérer la fenêtre en mettant le processus en "tâche de fond". Ceci peut se faire :

- soit au lancement de la commande qu'il suffit de terminer par &.
- soit au cours de l'exécution, en général un 'Ctrl-z' stoppe le processus et un *bg* le place en background (il est possible de rappeler le processus avec *fg*.

Si l'on souhaite que le processus placé en tâche de fond continue à s'exécuter après la mort du shell qui l'a lancé il faut alors précéder la commande d'un *nohup*, par exemple :
nohup truc.exe &

Redirection

Le shell permet de dupliquer, fermer et rediriger les canaux d'entrée sorties, en utilisant le codage standard suivant : Entrée standard (0), Sortie standard (1), Erreur standard (2).
On peut par exemple associer le canal 0 avec un autre canal *n* comme suit : "commande <& *n*".
On peut bien entendu faire de même avec le canal 1 : "commande >& *n*".
Pour fermer les canaux 0 et 1, on utilise les syntaxes suivantes :
"commande <&- " pour le canal 0
"commande >&- " pour le canal 1
On peut également associer aux caractères < et > un numéro de canal en le placant juste avant. On peut ainsi, rediriger l'erreur standard vers un fichier quelconque "2>", ou bien l'envoyer sur le même canal que la sortie standard "2>&1"

Arborescence

Les fichiers sont organisés en arborescence :

- la racine est le /
- chaque niveau est un répertoire
- pour chaque répertoire on crée :
  - une référence au répertoire courant •
  - une référence au répertoire père dans l'arbre ••
- lorsque l'on situe un point dans l'arbre,chaque niveau est séparé par un /
- une adresse dans l'arbre, peut être relative à l'endroit où l'on se trouve ou bien absolue, en partant de la racine.

Manipulation de répertoires

Il y a de nombreuses commandes pour manipuler les fichiers ou les répertoires, en voici quelques unes qui peuvent s'appliquer (entre autres) sur un répertoire :

Utilitaire	Origine	Description
pwd	Print Working Directory	répertoire courant
ls	List Saved	contenu d'un répertoire
cd	Change Directory	Change le répertoire
mv	Move	Déplace ou renomme
rmdir	Remove Directory	Détruit un répertoire
mkdir	Make Directory	Crée un répertoire
ln	Link	Crée un lien symbolique
chmod	Change Mode	Modifie les droits
chown	Change Owner	Change le possesseur
chgrp	Change Group	Change le groupe

commandes utiles

Voici quelques commandes utiles avec leurs options les plus courantes :

- "ls -al" : donne la liste de tous les fichiers du répertoire courant, en détail
- "ps aux" : donne la liste de tous les processus courant, en détail
- "df" : donne l'état de toutes les partitions actuellement en usage
- "du -sk truc" : indique en kilo octets la place occupée par un répertoire truc
- "ls -al |sort -n -k 5" : trie la liste des fichiers du répertoire en fonction de la place occupée

Les droits d'un fichier

La confidentialité des fichiers dans un système **Unix** repose sur un système de droits très simple. On distingue trois catégories d'utilisateurs, la personne qui possède le fichier (user), les membres du même groupe que celui du fichier (group), et enfin tout le monde (other). Il y a trois sortes de droits, lecture (r), écriture (w) et exécution (x) dont les effets sont les suivants :

- Lecture (r)
  Il permet de lire un fichier texte ou d'effectuer un "ls" sur un répertoire,
- Ecriture (w)
  Il permet de créer, modifier ou effacer un fichier,
- Exécution (x)
  Il permet d'exécuter un fichier ou de traverser un répertoire.

Droits spéciaux

Il existe des droits particuliers, importants, car leur mauvaise utilisation peut créer des problèmes de sécurité :

- set-uid ou set-gid (s) : la permission x sur un fichier peut être remplacée par la permission s que l'on peut affecter au propriétaire (suid) et/ou au groupe (sgid) du fichier. Cette permission indique que le fichier est exécutable et que pendant son exécution on prend les droits du propriétaire et/ou du groupe du fichier exécutable.
- sticky bit (t) : Sur un fichier exécutable, il indique que le code reste en mémoire aprés exécution. Sur un répertoire, il indique que seul le possesseur, ou root, peut détruire ce répertoire.

C'est bien entendu le privilège set-uid (ou set-gid) qui est le plus sensible, car il permet, dans le cas où le fichier appartient à root d'obtenir les privilèges root durant l'exécution.

Manipuler les droits

La commande **chmod** permet de modifier les droits d'un fichier, les droits peuvent être en octal (ex *chmod 755 truc*, donne les permissions `-rwxr-xr-x` au fichier truc), ou bien de façon symbolique :

- Pour les utilisateurs : **u** signifie possesseur, **g** signifie groupe, **o** signifie le reste du monde, **a** signifie les trois types précités.
- Pour les droits : les lettres **rwxst** définies précédemment.
- Pour les actions : **=** assigne des droits, **+** ajoute des droits, **-** supprime des droits.

Par exemple :

- *chmod a=rx truc* donne les droits `-r-xr-xr-x` au fichier.
- *chmod o+rw truc* rajoute les droits rw, si besoin est, pour le reste du monde.
- *chmod go-x truc* supprime le droit d'exécution pour le groupe et le reste du monde.

umask

La commande **umask** définit un masque décrivant les permissions des fichiers lors de leur création, c'est un complément à 1 de la permission désirée, il est exprimé en octal. Quelle que soit la valeur du masque, les "fichiers" ne sont jamais créés avec le droit x.
Le tableau 2 fournit les valeurs en octal correspondant aux droits des fichiers. Une fois le mécanisme compris, l'utilisation est simple, par exemple la commande *umask 066* positionne le masque de création des fichiers à la valeur octale 066. Par la suite les répertoires seront créés avec les permissions `drwx-x-x` et les fichiers avec les permissions `-rw-----`.

Droit et masque octal

Les permissions		
Droits symboliques	Valeur octale	Masque
`rwx`	7	0
`r-`	4	3
`-w-`	2	5
`-x`	1	6
`rw-`	6	1
`r-x`	5	2
`-wx`	3	4

TABLE : Les droits, leur représentation en octal et la valeur du masque correspondant

Fichiers de configuration

La plupart des applications utilisent des fichiers de configuration, placés dans le répertoire de travail de l'utilisateur, ces fichiers commencent généralement par un point :
.mailrc, .cshrc, . kshrc, .profile, .emacs, .dtprofile, ...
Ils ont la particularité de rester invisibles si on regarde le contenu du répertoire avec une commande *ls* simple et n'apparaissent que si l'on rajoute l'option *-a* (all). Mais ce sont presque toujours des fichiers textes que l'on peut modifier à loisir (pas toujours sans conséquences ... ;-0)).
Souvent les applications créent et gèrent elles même leur fichier de configuration épargnant aux utilisateurs le casse tête du paramêtrage.

Constructions des noms de fichiers

Il n'est pas toujours nécessaire de connaître le nom complet des fichiers que l'on veut traiter. Il existe des caractères spéciaux que le shell utilise pour générer un ou plusieurs noms de fichiers.

- * remplace n'importe qu'elle chaîne de caractères (y compris la chaîne vide),
- ? remplace un caractère quelconque,
- [ ] remplace les caractères entre crochets, par enumération ou intervalle (à l'aide du -). Un caractère ! dans les crochets, signifie la négation.

par exemple *ls -al *.c* n'effectue le *ls -al* que sur les fichiers se terminant pas *.c*.

cp $[a-cA-C]*$ *../Old*, copie tous les fichiers commençant par a,b,c,A,B,C dans le répertoire *../Old*.

Aide

Quelques moyens pour obtenir de l'aide :

help

- la commande man affiche le manuel d'une commande (quand il existe ;-)),
- la commande info navigue au sein de la documentation d'une commande (si elle existe ;-)) ,
- il existe un répertoire /usr/share/doc contenant de la documentation,
- on peut trouver des HowTo ou des FAQ sur certains sujets,
- certaines commandes listent leurs options lorsqu'elles sont appelées avec le paramêtre -h , -help ou –help,
- google est votre ami :-)

Différentes utilisation de bash

Le shell peut être utilisé de plusieurs façons différentes :

Utilisation

- shell de login
- shell interactif
- exécution d'un script (non-interactif)

Sous Linux, lorsqu'on lance un sh (bourne-shell), en réalité on exécute un bash dans un mode particulier, où celui-ci copie le comportement du sh.
Un shell peut parfaitement être de login et intéractif.
Selon les cas, la séquence d'initialisation sera différente.

Séquence d'initialisation de bash

Le bash, bourne again shell, est comme son nom l'indique, un descendant du bourne-shell. Dans le cas de l'utilisation en mode interactif, son démarrage se déroule comme suit :

interactif login

- exécution du fichier "/etc/profile" s'il existe,
- exécution du premier fichier trouvé dans le répertoire de l'utilisateur, parmi ".bash_profile", ".bash_login", ".profile".

Dans le cas où il s'agit d'un shell interactif mais pas de login :

interactif non-login

- exécution du fichier "/etc/bash.bashrc" s'il existe,
- exécution du fichier ".bashrc" de l'utilisateur s'il existe.

Variables du shell

Le shell possède deux sortes de variables : des variables locales et des variables d'environnement.

Déclaration

- Pour créer ou modifier une variable locale dénommée *truc*, il suffit de faire *truc=valeur*
- Il est possible d'exporter la variable locale *truc* en variable d'environnement en faisant *export truc*
- On peut également utiliser *export* directement pour créer une variable d'environnement.

Manipulations sophistiquées

Manipulation des variables	
$*nom_variable*	la valeur de *nom_variable*
${*nom_variable-mot*}	*nom_variable*, si elle existe, sinon par *mot*.
${*nom_variable+mot*}	*mot* si *nom_variable* existe, sinon par rien du tout.
${*nom_variable=mot*}	*nom_variable*, en lui donnant la valeur *mot* si elle n'existait pas.
${*nom_variable ?mot*}	*nom_variable*, si elle existe, sinon *mot* est envoyé sur le canal d'erreur et le shell est terminé, Si on omet *mot* un message standard est envoyé à la place.

Personnalisation

Avant même de chercher à programmer en shell-scripts, il est utile de savoir personnaliser son environnement, en modifiant quelque peu son fichier ".bashrc".

Alias

En utilisant la commande *alias*, il est possible de se construire des "macros"-commandes par exemple :
alias lsort='ls -al |sort -n -k 5'
Il est bien entendu possible de défaire cet alias grace à *unalias*

Path

Lorsque l'on entre une commande, le shell parcourt une succession de répertoires et exécute la première instance trouvée.
La variable PATH contient ces répertoires séparés par des ' :'.

Environnement

Autres variables

- HOME contient l'adresse absolue du répertoire de l'utilisateur,
- MANPATH contient les répertoires où se trouvent les manuels,
- LD_LIBRARY_PATH contient les répertoires où se trouvent les librairies dynamiques,
- TERM contient le type de terminal utilisé,
- MAIL contient l'adresse absolue de la boîte aux lettres,
- USER contient l'identificateur de l'utilisateur,
- PS1 contient le prompt affiché en début de ligne (il existe un certain nombre d'options permettant de le configurer),
- PWD contient l'adresse absolue du répertoire courant.

Caractères spéciaux

un shell-script est un fichier qui va être interprété lors de son exécution par le shell.

- Tout ce qui suit # est un commentaire,
- Sur la première ligne # ! indique le chemin de l'interpréteur,
- Les mots de la ligne de commande sont placés dans les variables $0, ..., $9 (y compris le nom du script)
- $# est le nombre d'arguments (non compris le nom du script)
- $* et $@ la liste des arguments mais
  - "$*" est équivalent à "$1 $2 ..."
  - "$@" est équivalent à "$1" "$2" ...

Structures de contrôle

Le shell possède un assortiment complet de structures de contrôle :

boucles et autres

- **for .. do .. done**
  la boucle la plus utilisée,
- **until .. do .. done**
- **while .. do .. done**
- **if .. then .. elif .. then .. else .. fi**
  elif et else sont optionnels, en général on a seulement **if .. then .. fi**
- **case .. in .. esac**
- **break** termine immédiatement la structure
- **continue** passe immédiatement à l'itération suivante

Exemple

Voici un petit exemple de boucle avec **for .. do .. done** :

```
for i in a b c d e f
do
  mkdir i
done
```

La boucle précédente, crée les répertoires a/ b/ c/ d/ e/ f/ dans le répertoire courant.
Attention : le shell manipule des chaînes de caractères par défaut et non des nombres.

Arithmétique

exp

Pour faire des calculs il faut utiliser la commande **expr**.

- **expr** *expression*

Evalue *expression* et l'envoie sur la sortie standard.

opérateurs arithmétiques

Opérateur	Signification
* / %	multiplication, division ,modulo
+ -	addition, soustraction
= \ > \ >= \ < \ <= ! =	Comparaisons
\&	et logique
|	ou logique

Les parenthèses \(et \) doivent être utilisées lorsque l'on veut changer l'ordre du calcul.

case .. in .. esac

case

La commande **case** permet des branchements multiples basés sur la seule valeur d'une chaîne de caractères.

case *expression* **in**
motif {|*motif*}*)
    liste
    ;;
esac

for .. do .. done

for

La boucle **for** permet à une liste de commandes d'être exécutée de nombreuses fois en utilisant une valeur différente de la variable de boucle à chaque itération.

for *nom* [**in** {*mot*}*]
do
    liste
done

if .. then .. fi

if

La structure **if** permet des branchement conditionnels.

```
if liste1
then
    liste2
elif liste3
then
    liste4
else
    liste5
fi
```

if (suite)

détails

La partie **elif** est optionnelle et peut être répétée.
La partie **else** est optionnelle et peut intervenir zéro ou une fois.

- Les commandes dans *liste1* s'exécutent.
- Si la dernière commande de *liste1* a réussi on exécute *liste2*.
- Si la dernière commande de *liste1* est un échec et qu'il y a un ou plusieurs **elif**, une *liste3* réussie provoque l'exécution du *liste4* qui suit le **then**.
- Si on ne trouve pas de liste réussie et qu'il y a un **else** alors les commandes *liste5* suivant le *else* sont exécutées.

until .. do .. done

until

La commande **until** exécute une série de commandes aussi longtemps qu'une autre série de commandes échoue.

until *liste1*
do
    liste2
done

On exécute *liste1* et on termine si la dernière commande de *liste1* réussit. Autrement les commandes de *liste2* sont exécutées et le processus est répété. Si *liste2* est vide on peut omettre **do**.

while .. done

while

La commande **while** exécute une série de commandes aussi longtemps qu'une autre série de commandes réussit.

while *liste1*
do
    liste2
done

On exécute *liste1* et on termine si la dernière commande échoue. Sinon on exécute les commandes de *liste2* et le processus est itéré. Les mêmes remarques que pour **until** concernant le **do**.

test

Expressions conditionnelles

La commande **test** permet de tester une expression :
test *expression* ou bien **[** expression **]**
Retourne une valeur de sortie égale à 0 si l'expression est vraie et une valeur non nulle sinon.

comparaisons

Forme	**Signification**
entier1 -eq *entier2*	Vraie si égalité
entier1 -ne *entier2*	Vraie si différents
entier1 -gt *entier2*	Vraie si plus grand
entier1 -ge *entier2*	Vraie si plus grand ou égal
entier1 -lt *entier2*	Vraie si plus petit
entier1 -le *entier2*	Vraie si plus petit ou égal

test (suite)

Il est possible de tester l'existence et la propriété de fichiers :

fichiers

Forme	Signification
-b *nom de fichier*	Vraie si fichier bloc
-c *nom de fichier*	Vraie si fichier caractère
-d *nom de fichier*	Vraie si répertoire
-f *nom de fichier*	Vraie si non répertoire
-g *nom de fichier*	Vraie si fichier "set-group-id"
-h *nom de fichier*	Vraie si lien symbolique
-k *nom de fichier*	Vraie si possède le sticky bit
-u *nom de fichier*	Vraie si fichier "set-user-id"
-w *nom de fichier*	Vraie si fichier dans lequel on peut écrire
-x *nom de fichier*	Vraie si fichier exécutable

Gérer des signaux

trap

Cette commande permet d'exécuter certaines commandes lors de la réception de signaux particuliers, comme suit :

trap [[*commande*]{*signal*}+]

Cette commande indique au shell d'exécuter *commande* chaque fois qu'un des signaux énuméré de façon numérique dans *signal* est reçu. Si plusieurs signaux sont reçus, ils sont traités dans l'ordre croissant. Si 0 est spécifié la *commande* s'exécutera à la terminaison du shell.

Chercher des fichiers

find

La commande find, permet de rechercher des fichiers selon de nombreux critères puis, au besoin, d'effectuer certaines actions sur ces mêmes fichiers. Il faut donc au minimum :

- un point de départ dans l'arborescence
- un critère de recherche

Elle permet également d'exécuter des commandes sur les fichiers trouvés.

Exemples de "find"

Exemples

- Chercher un fichier récursivement à partir du répertoire courant

```
find . -name nom_fichier
```

- Compresser avec gzip tous les fichiers pdf (récursivement) à partir du répertoire courant

```
find . -iname "*.pdf" -exec gzip -f {} \;
```

- Trouver tous les fichiers de taille supèrieure à 1Mo à partir du répertoire courant et exécuter `ls -l`

```
find . -size +1M -exec ls -l {} \;
```

Chercher dans un fichier

grep

La commande grep permet de rechercher des mots ou des expressions régulières dans des fichiers texte. Elle possède de nombreuses options dont la possibilité de rechercher récursivement (*-r*)

Exemples

- Rechercher les occurences du mot `pas` dans tous les fichiers commençant par `truc`

```
grep pas truc*
```

- Rechercher récursivement à partir du répertoire courant les occurences de chaînes de caractères comportant *#!* et *sh*, (les scripts shell en fait)

```
grep -r "#\! /[0-9A-Za-z/]*sh" *
```