UNIX : Compléments, trucs et astuces

Gérald MONARD

DESS MOIC - Année 2002-2003

Bibliographie: Nebut, Unix Power tools

Contents

Chapter 1

Présentation Générale et Manipulations de base

1.1 Présentation Générale d'UNIX

UNIX a été inventé (écrit) par Kenneth Thompson et Dennis Ritchie au début des années 1970 au sein des laboratoires Bell (AT&T).

Ont droit à la mention "UNIX" les systèmes d'exploitation validés par l'OSF (Open Software Foundation). Linux est seulement un "UNIX-like" car il n'a pas la certification (payante) OSF.

Il existe deux grandes variantes d'UNIX : *System V* et *BSD*.

1.1.1 Caractéristiques essentielles

Le système UNIX set organisé en couches

1. les pilotes qui réalisent les échanges avec les périphériques et les opérations élémentaires pour gérer la mémoire ;
2. le système de fichiers (incluant la gestion des entrées-sorties) et le système de gestion des tâches (incluant la gestion de la mémoire) ;
3. les différents outils accessibles par des commandes.

Les deux premières couches représentent le *noyau* du système.

UNIX est *multi-tâches* : plusieurs tâches peuvent s'exécuter logiquement en parallèle à un instant donné. Il est *multi-utilisateurs* : pour pouvoir travaillé sur un système UNIX, un utilisateur doit être enregistré, et plusieurs utilisateurs peuvent travailler simultanénament.

Les entrées-sorties sont banalisées : écrire sur un fichier, sur un périphérique, sur une prise réseau ou sur l'entrée d'une tâche se fait toujours d'une façon identique. Cette unification est réalisée par le système de fichiers, et les différences sont traitées au dernier moment, par la couche la plus interne du noyau.

1.1.2 Classement des Utilisateurs

Il y a deux sortes d'utilisateurs

- l'*administrateur* du système : c'est un usage particulier qui a tous les droits, et qui est responsable du bon état du système. Son nom est **`root`**.
- Les autres utilisateurs. C'est l'administrateur qui enregistre un nouvel usager. Lorsqu'un usager se connecte sous Unix, il ouvre une *session*. Les usagers sont regroupés en groupes.

1.2 Les Processus

Une tâche est appelée un *processus* sous Unix. Un processus est une activité autonome qui exécute un programme : c'est une entité dynamique, qui naît, qui vit en toute indépendance ou en communicant avec d'autres processus, qui à son tour peut créer des processus, et enfin qui meurt.

Un processus est caractérisé par un certain nombre d'informations stockées dans une table du noyau dont :

- son numéro (PID)
- le numéro de son père (PPID)
- son propriétaire (UID)
- son groupe propriéaire (GID)

1.2.1 Voir les processus en cours

La commande **`ps`** affiche divers renseignements sur les processus actifs du système.

Sans option, **`ps`** donne la liste des processus attachés au *terminal*.

`ps` sous *System V*

-a tous les processus attachés à un terminal

-e tous les processus

-f plus de renseignements (*full*)

-l plus de renseignements (*long*)

-pn[,n] les processus de numéro(s) n

-uid[,id] les processus des utilisateurs de numéros id

`ps` sous *BSD*

-a tous les processus détenus par l'utilisateur

-r tous les processus qui ne sont pas attachés à un terminal

-u usager

-l plus de renseignements

1.2.2 Tuer des processus

`kill` sert à tuer un (ou plusieurs) processus connaissant son numéro d'identification (PID).

`kill` [*-n*] [*numéro*]

n est le numéro de signal envoyé au processus. Par défaut, le signal 15 est envoyé (terminate). 9 est un signal qui tue à coup sûr (en principe !).

1.2.3 Abaisser la priorité d'un processus

L'administrateur peut augmenter la priorité de n'importe quel processus. Un utilisateur lambda ne peut lui que l'abaisser. La commande **`nice`** lance une commande avec une priorité donnée.

`nice` [*-priorité*] *commande*

priorité est un entier compris entre 0 (minimum) et 19.

1.2.4 Action sur un processus

Un processus peut être exécuté de manière asynchrone en utilisant la commande **&**.

Un processus peut être stoppé par **`Ctrl-Z`**. Il peut être mis en arrière plan par la commande **`bg`** (background), où être remis au premier plan par la commande **`fg`**.

Dans le cas d'un processus stoppé ou en arrière plan, celui-ci peut être tué par la commande **`kill %`**.

1.3 Le Système de Fichiers

Un fichier est une suite non structurée de caractères, stockée sur une mémoire auxiliaire. Le système fournit deux modes d'accès : l'accès séquentiel, pour accéder au caractère suivant, et l'accès direct, pour accéder au caractère de numéro donné.

Il existe différentes sortes de fichiers :

- les fichiers ordinaires servant à stocker des programmes ou des données ;
- les répertoires

- les périphériques à accès par caractère (ex.: un terminal)
- les périphériques à accès par blocs (ex.: un disque)
- les fichiers à références vers un autre volume (*i.e.*, les liens symboliques)
- les fichiers servant de tubes mettant en communication deux processus
- les fichiers qui sont des prises réseaux, pour accéder à des fichiers éloignés.

1.3.1 Caractéristiques d'un fichier

Les caractéristiques statiques d'un fichier (celles qui ne dépendent par de son accès à un instant donné) sont stockées dans un descripteur appelé *i-nœud*. Un i-nœud est un tableau qui contient les informations suivantes :

- le type de fichier (ordinaire, répertoire, ...)
- sa taille (en octets)
- l'identification du propriétaire (UID-GID)
- les droits/protections d'accès sur ce fichier
- trois dates : la date de création, la date de dernière modification et la date de dernière consultation
- un compteur de références sur le i-nœud
- la liste des blocs contenant l'information sur le disque

1.4 Besoin d'aide ?

La majorité des commandes UNIX possède une documentation interne accessible par la commande **`man`**. Sur les systèmes Linux, des informations aussi peuvent être obtenues via la commande **`info`**.

1.5 Manipulation des fichiers

1.5.1 Répertoire

Lorsqu'on ouvre une session, un utilisateur est placé à un endroit dans l'arborescence des fichiers et des répertoires.

La commande **`pwd`** permet de connaître sa position dans l'arborescence. La création d'un répertoire s'effectue à l'aide de la commande **`mkdir`**. La suppression d'un répertoire (vide) s'effectue grâce à la commande **`rmdir`**.

. représente le répertoire courant.

.. représente le répertoire père.

1.5.2 Contenu d'un répertoire, attributs d'un fichier

La commande **ls** permet de visualiser la plupart des attributs d'un fichier ou des fichiers donnés en paramètre, et également d'afficher le contenu d'un répertoire.

ls [*options*] [*noms*]

Quand *nom* est un nom de répertoire, **ls** affiche la liste des noms de fichiers ou de répertoires qu'il contient, triés par ordre alphabétique. Quand *nom* est absent, le répertoire . ($CWD) est pris par défaut.

Options :

-a affiche tous les noms de fichiers, même ceux commençant par un point

-i affiche le numéro de i-nœud devant chaque nom de fichier ou de répertoire

-l affichage long :

- nature du fichier : **-** pour un fichier, **d** pour un répertoire (*directory*), **l** pour un lien symbolique, **c** pour un périphérique en mode caractère, **b** pour un périphérique en mode bloc, **s** pour une prise réseau, **p** pour un tube nommé.
- les droits d'accès pour le propriétaire, le groupe et les autres utilisateurs : **r** droit en lecture, **w** droit en écriture, **x** droit en exécution, **-** pas de droit.
- le nombre de liens du fichier (1 veut dire un seul nom)
- les noms du propriétaire et du groupe
- la taille du fichier en octet
- la date de dernière modification
- le nom du fichier

-d pour un répertoire, affiche le nom ou les attributs du répertoire plutôt que le contenu

-t affichage trié par la date de dernière modification

-F affichage de '/' après le nom d'un répertoire, '*' après le nom d'un exécutable

1.5.3 Occupation des fichiers

La commande **du** permet de connaître l'occupation sur le disque d'un fichier ou d'un répertoire

du [*option(s)*] [*fichier(s)*]

Sans option, **du** donne le nombre de quotas[1] de chaque répertoire de la sous-arborescence définie par le répertoire courant (récursivement pour chaque sous-arborescence).

Options :

[1] 1 quota vaut 512 octets sous System V et 1024 octets sous BSD

-k donne les quotas en kilo-octets

-s n'affiche le nombre de quotas que pour les fichiers nommés en argument

-a donne le détail au niveau de chaque fichier (et non de chaque répertoire) avec le récapitulatif par répertoire

1.5.4 Propriétaire et droit d'accès

Les protections en lecture, écriture et exécution sont placées dans le i-nœud d'un fichier lors de sa création sous la forme d'un ensemble de neuf bits : 0 = droit absent, 1 = droit présent. Ces neuf bits sont regroupés sous la forme de trois chiffres octaux. Un chiffre octal correspond aux trois bits rwx (dans cet ordre). Le premier octet correspond aux droits du propriétaire, le deuxième aux droits du groupe et le troisième aux droits des autres.

Droits par défaut

Les droits par défaut sont définis par un masque de création, établi par la commande **`umask`**.

Sans paramètre, **`umask`** affiche la valeur du masque courant.

On obtient les droits par défaut d'un fichier par l'opération :

666 **et** (**non** masque)

On obtient les droits par défaut d'un répertoire par l'opération :

777 **et** (**non** masque)

`umask` *masque* permet de changer la valeur du masque.

Changement des droits

Les droits établis à la création d'un fichier peuvent être modifiés par la suite par la commande **`chmod`**

`chmod` [*option(s)*] *mode* *fichier(s)*

mode peut prendre deux formes :

- soit trois chiffres octaux
- soit [*qui*] *position* [*droit*] où

  qui est l'un des caractères suivant : **`u`** pour le propriétaire, **`g`** pour le groupe, **`o`** pour les autres, **`a`** pour tous. Il indique pour qui on donne les droits qui suivent (défaut : **`a`**)

  position est l'un des caractères suivant : **`+`** pour ajouter un droit, **`-`** pour retirer un droit, **`=`** pour mettre un droit et supprimer les autres

droit est l'un des caractères suivant : **r** pour le droit **r**, **w** pour le droit **w**, **x** pour le droit **x**

Options :

-R récursif

Changement de propriétaire

Le propriétaire d'un fichier peut changer le groupe d'un fichier avec la commande **chgrp** :

chgrp *groupe* *fichier(s)*

Il peut également changer le propriétaire d'un fichier avec la commande **chown** :

chown *nom* *fichier(s)*

Changement des dates

touch *option(s)* *fichie(s)*

touch crée les fichiers s'ils n'existent pas, ou met à la date du jour les dates de dernière consultation et de dernière modification des fichiers.

Options :

-a ne modifie que la dernière date de consultation (accès)

-m ne modifie que la dernière date de modification

-c ne crée pas le fichier s'il n'existe pas

-r *reffile* utilise la date de *reffile* comme valeur de date plutôt que la date courante

-t *time* utilise le temps *time* comme valeur de date plutôt que la date courante

1.5.5 Copie, déplacement, renommage, destruction

Copie d'un ou plusieurs fichiers

La commande :

cp *origine* *resultat*

copie le fichier *origine* dans le fichier *resultat*.

Si *resultat* est un répertoire, le fichier *origine* est recopié dans le répertoire *resultat*.

Options :

-r copie récursive

-p préserve les droits et les dates de dernière accès et de dernière modification

Déplacement ou renommage

mv *ancien nouveau*

Liens

La commande :

ln *nom autrenom*

permet d'ajouter une référence *autrenom* au fichier *nom*.

Dans le cas où *autrenom* est un nom de répertoire déjà existant :

ln *fichier(s) repertoire*

des liens pour chaque *fichier* sont créés dans *répertoire*.

Options :

-s lien symbolique (ne contient que la référence du chemin et non le contenu)

Destruction d'un fichier

rm [*option*] *fichier(s)*

Options :

-r récursif

-f force la destruction sans demander à l'utilisateur

-i destruction interactive

1.5.6 Affichage interactif

more est un filtre qui permet l'affichage page par page d'un fichier texte, avec une pause en fin de chaque page. Après une pause, on peut reprendre le défilement, appeler l'éditeur **vi**, ...

more *fichier(s)*

Requêtes :

entrée affiche la ligne suivante

espace affiche la page suivante

q quitter

/ *chaîne* recherche le motif *chaîne*

less est un filtre d'affichage par page similaire à **more**, mais il permet les remontées dans les fichiers visualisées.

1.5.7 Longueur d'un fichier

wc affiche le nombre de caractères, de mots et de lignes d'un fichier. Options :

-l n'affiche que le nombre de lignes

-w n'affiche que le nombre de mots

-c n'affiche que le nombre de caractères

1.5.8 Affichage complet, par pages, du début ou de la fin d'un fichier

cat lit des fichiers et les écrits sur la sortie standard : il fait donc un affichage de fichiers, et en même temps, les concatène.

head affiche les premières lignes d'un fichier.

head [-*n*] *f*

Par défaut, les 10 premières lignes de *f* sont affichés, ou les *n* premières.

tail affiche les dernières lignes d'un fichier.

tail [-*n*] [-*f*] *fichier*

Par défaut, les 10 dernières lignes de *fichier* sont affichés, ou les *n* premières si demandées.

L'option *-f* permet d'afficher les lignes au fur et à mesure où elles sont ajoutées au fichier.

1.5.9 Sauvegarde de la sortie standard

La commande **tee** copie son entrée standard sur la sortie standard et dans un ou plusieurs fichiers donnés en paramètre.

tee [-*a*] *fichier(s)*

Options :

-a ajoute en fin de fichier

Cette commande est principalement utilisée comme filtre de la sortie standard.

1.5.10 nom et répertoire de location d'un fichier

basename permet d'extraire des portions de noms d'un fichier : elle retire les portions du nom qui sont des noms de répertoires ainsi que le suffixe, s'il est précisé.

dirname délivre tous les composants qui sont des répertoires du *nom*.

basename *nom* [*suffixe*]

dirname *nom*

1.5.11 Comparaison de fichiers

sans détail

cmp fait une comparaison rapide de deux fichiers :

cmp *fichier1 fichier2*

et retourne le code retour 0 si les fichiers sont identiques, plus un commentaire. L'option *-s* supprime ce commentaire.

avec détail

diff fait également une comparaison (ligne à ligne) de deux fichiers, mais en précisant quelles sont les lignes qui ne sont pas identiques.

1.5.12 Compression de fichiers

compress

compress *fichier(s)*

La commande **compress** permet de compresser un fichier dans le but de préserver de l'espace disque.

fichier est alors transformé en *fichier.Z*

Pour retransformer *fichier.Z* en *fichier*, on utilise la commande **uncompress**.

L'équivalent de **cat** pour un fichier compressé est **zcat**.

compress -c équivaut à **zcat**.

gzip

gzip *fichier(s)*

Même chose que **compress** mais en plus efficace. *fichier* est alors transformé en *fichier.gz*

Autres commande : **gunzip**, **zcat**.

bzip2

bzip2 *fichier(s)*

Même chose que **gzip** mais en plus efficace. *fichier* est alors transformé en *fichier.bz2*

Autres commande : **bunzip2**, **bzcat**.

1.5.13 Archivage

ar

ar permet de construire des bibliothèques de programmes objet pour l'éditeur de liens, ou d'archiver dans un seul fichier plusieurs programmes source.

ar *clé archive* [*fichier(s)*]

archive est le fichier d'archivage, créé à la première commande d'archivage; son nom est habituellement suffixé par *.a*.

clé est un code spécifiant l'opération souhaitée sur l'archive :

t imprimer la table du contenu de l'archive

r ajouter des fichiers dans l'archive, ou les remplacer

x sortir les fichiers de l'archive

d détruire les fichiers de l'archive

p afficher le contenu des fichiers de l'archive

Si l'archive contient des modules objet, elle devient une bibliothèque exploitable par l'éditeur de liens **ld**. Si l'archive ne contient que des textes, elle est imprimable (par **cat**, **more**, ...).

tar

tar *clé* [*fichier(s)*]

tar sert à archiver (ou extraire) des fichiers ou des répertoires sur bandes, disques souples, cartouches, ...

fichier est un fichier à archiver sur (ou à extraire d') un support externe. Si c'est un répertoire, cela implique l'archivage (ou l'extraction) des fichiers du répertoire, et récursivement pour les sous-répertoires.

clé est un code spécifiant l'opération souhaitée sur l'archive :

-t imprimer les noms des fichiers de l'archive

-r ajouter des fichiers à l'archive

-x extraire les fichiers de l'archive

-c créer l'archive à partir du début de l'archive

Options possibles :

v "verbeux"

f l'argument qui suit cette option est le nom de l'archive. Si c'est un tiret, c'est l'entrée ou la sortie standard qui est impliquée.

C implique un changement de répertoire

1.5.14 Recherches

which

`which` permet de connaître le chemin complet d'une commande (parcourt *$PATH*).

Options :

-a permet de connaître tous les chemins possibles et non le premier trouvé

-i permet de lire aussi les alias (**`alias | which -i`**)

find

`find` descend récursivement des sous-hiérarchies de répertoires données par leur racine, en cherchant à appliquer à des fichiers précisés par un ou plusieurs critères de sélection (nom, type, date, ...) une commande donnée.

`find` [*répertoire*] [*expression*]

répertoire est la liste des racines des sous-hiérarchies à parcourir. *expression* est une suite d'options exprimant à la fois des critères de sélection des fichiers et les actions à leur appliquer. Lorsque le critère est vrai, l'action est exécutée. Dans la suite, on appelle "fichier courant" le fichier examiné par **`find`** à un instant donné.

-name *motif* vrai si le motif s'applique sur le nom du fichier courant

-user *nom* vrai si le fichier courant appartient à l'utilisateur *nom*

-atime *n* vrai si le fichier a été utilisé dans les *n* derniers jours. (Peu utilisable car **`find`** modifie la date d'accès aux fichiers qu'il examine)

-mtime *n* vrai si le fichier a été modifié dans les *n* derniers jours

-newer *fich* vrai si le fichier courant a été modifié plus récemment que *fich*

(*expression*) vrai si l'expression est vraie

-print toujours vrai, affiche le nom du fichier courant

-exec *com* vrai si la commande *com* délivre un code de retour nul. *com* est terminé par le marqueur `\;`, et le paramètre spécial { } est la notation qui désigne le fichier courant

Les éléments de l'expression peuvent être connectés par les opérateurs suivants, donnés dans l'ordre de priorité décroissante :

- l’opérateur de négation : le point d’exclamation `!`
- l’opérateur ”et” : simple juxtaposition des éléments
- l’opérateur ”ou” : `-o`

Chapter 2

Shell, script

Sous UNIX, l'interprêteur qui entoure virtuellement l'ensemble des commandes est appelé un shell. Le langage de commande interprêté n'est pas unique, on peut choisir un langage parmi plusieurs : Bourne-Shell, C-Shell, Korn-Shell, Bash, ...

2.1 Niveau lexical

2.1.1 Séparateurs

<**Return**> termine une ligne de commande

; idem

2.1.2 Caractères génériques

? remplace un caractère quelconque

***** remplace n'importe quelle chaîne, même vide, de caractères ne comportant pas de `<Return>`

[] remplace un caractère parmi ceux qui sont énumérés entre les crochets (ou un caractère parmi ceux qui ne sont pas énumérés entre les crochets si le premier d'entre eux est un point d'exclamation).

{} permet de factoriser une liste de noms de fichiers

2.1.3 Caractères spéciaux

Le caractère de continuation \ placé en fin de ligne indique que la phrase en cours n'est pas terminée.

Les caractères spéciaux ont une signification dans le langage, à moins qu'ils ne soient ignorés par l'emploi du caractère \ en précédence.

2.1.4 Parenthésages

Les parenthésages servent à regrouper des séquences de mots ou à isoler des mots :

(…) entoure une liste de commandes qui sont exécutées par un nouveau processus UNIX

'…' (quotes) parenthésage dans lequel aucun caractère sauf ' n'a de signification particulière

"…" (guillemets) parenthésage dans lequel aucun caractère sauf ‘, " et $ n'a de signification spéciale

‘…‘ (accents graves) exécution de la phrase placée entre accents graves, délivre la sortie standard des commandes situées dans ce parenthésage.

2.2 historique

La commande **`history`** permet d'afficher les commandes précédentes (mises en tampon). Chaque commande est repérée par un numéro dans le tampon (historique des commandes).

Référence à une commande dans l'historique

$!<num>$ rappelle la référence $<num>$

$!-n$ rappelle la n-ième plus récente commande

!! rappelle la dernière commande

$!x$ rappelle la dernière commande commançant par la chaîne x

$!?chaine?$ rappelle la dernière commande comportant quelque part la chaîne indiquée

Substitution dans les commandes de l'historique

Toute ligne de commandes est composée de mots, numérotés à partir de 0 (le nom de la commande).

Pour sélectionner un ou plusieurs mots d'une commande, on fait suivre la référence à la commande d'un deux points (:) et de la désignation des mots choisis :

n désigne le n-ième mot de la commande (0 désigne la commande)

$ désigne le dernier mot

***** désigne tout la commande, sauf le nom de la commande

Après chaque sélection de mot, on peut placer un modificateur précédé encore d'un deux points …

2.3 Abréviations

Une abréviation d'une commande est appelée un *alias* en C-Shell. Créer et détruire une abréviation s'effectuent par les commandes internes **`alias`** et **`unalias`**.

alias donne la liste des abréviations déjà définies

alias nom donne l'abréviation définie par l'alias *nom*

alias nom liste de mots crée l'abréviation *nom* pour *liste de mots*

unalias motif détruit le ou les alias dont les noms sont conformes au *motif* (mot comportant des caractères génériques)

Le caractère \ placé devant un alias inhibe le mécanisme de substitution des alias.

Les commandes d'abréviation peuvent être paramétrées en utilisant *!:numero*

2.4 Variables

Les variables sont des variables simples ou des variables tableaux. Leurs valeurs sont de type chaîne, mais certains opérateurs interprêtent ces chaînes comme des nombres ou des booléens.

2.4.1 Création, initialisation, destruction

Ces trois opérations sont réalisées par les commandes **`set`**, **`unset`** et **`@`**

set *nom* la variable *nom* est créée et initialisée à la chaîne vide

set *nom = val* la variable *nom* prend la valeur *val*

set *nom[ind] = val* la variable *nom* est un tableau et la variable indicée *nom[ind]* est initialisée avec *val*

set *nom = (liste de mots)* initialisation du tableau *nom*, chaque *mot* étant la valeur d'un élément du tableau

@ nom = expression affecte l'*expression* à la variable entière *nom*. (autre opérateur d'affectation : +=, -=, *-, /=, %=)

@ nom[ind] = expression idem pour une variable indicée

unset *motif* détruit la ou les variables satisfaisant *motif*

set, @ sans paramètre, ces deux commandes listent les variables connues

Les variables de l'environnement sont manipulées par la commande **`setenv`**

setenv *[nom [val]]* fonctionne comme les différentes formes de **`set`**, sans le signe =.

2.4.2 Expansion des variables

La valeur d'une variable (son "expansion" en la chaîne qui est sa valeur) est obtenue par l'opérateur $. La valeur obtenue peut être éventuellement un nom de fichier, une commande, ...

Si nécessaire, le nom de la variable est placé entre accolades.

La valeur des variables indicées est obtenue par la notation *$nom[ind]* ou *${nom[ind]}*, l'indice pouvant être lui-même une variable.

L'indice est un entier ou deux entiers séparés par un tiret pour accéder à une tranche du tableau.

$#nom ou *${#nom}* est le nombre d'éléments dans la variable *nom*

L'expansion d'une variable peut être modifiée en utilisant un ou plusieurs modificateurs, chacun précédé par un deux points (:)

:r retire l'extension .xxx

:e retire tout sauf l'extension

:h retire la composante d'un chemin, pour ne laisser que la "tête" (head)

:t retire la composante d'un chemin, pour ne laisser que la "queue" (tail)

:s/l/r/ substitue la chaîne 'l' par la chaîne 'r' (pas d'expression régulière)

2.4.3 Quelques variables connues de C-Shell

status le code de retour de la dernière commande

argv variable tableau contenant les paramètres effectifs passés à l'appel du C-Shell

cwd le répertoire courant

$ le numéro de processus en cours

2.5 Expressions

L'état d'un fichier peut être déterminé par une expression booléenne de la forme :

[-tt] *fichier*

dans laquelle le type de test **`-tt`** a comme valeur :

-e le fichier existe-t'il ?

-f est-ce un fichier ordinaire ?

-d est-ce un répertoire ?

-o en suis-je propriétaire ?

-z est-il de taille nulle ?

-r a-t'il les accès en lecture ?

-w en écriture ?

-x en exécution ?

2.6 Commandes

2.6.1 Commande simple

Une commande simple est une liste de mots séparés par des espaces.

Le premier mot est le nom de la commande, les autres sont les paramètres effectifs qui lui sont transmis.

Une valeur simple délivre une valeur appelée code de retour. Un code de retour nul indique une exécution sans erreur.

2.6.2 Pipeline

Un pipeline est une séquence de deux ou plusieurs commandes séparées par le caractère **|** (pipe). Les commandes sont exécutées par des processus différents fonctionnant en mode pipeline : la sortie standard du processus situé à gauche est connectée par un tube à l'entrée standard du processus situé à droite.

2.6.3 Liste de commandes

Une liste de commandes est une séquence de commandes simples ou de pipelines séparés par l'un des caractères suivants :

; exécution séquentielle

& exécution asynchrone

&& exécution de la partie droite conditionnée par un code de retour nul de ce qui précède (la commande s'est bien passée)

|| exécution de la suite conditionnée par un code de retour non nul de ce qui précède

2.6.4 Instructions ou commandes structurées

Boucle pour

`foreach` *indice* (*liste de valeurs séparées par des blancs*)
liste de commandes
`end`

Boucle tant que

`while` (*expression*)
    liste de commandes
`end`

Sélection si

`if` (*expression*) **`then`**
    liste de commandes
[**`else if`** (*expression*) **`then`**
    liste de commandes]
[**`else`**
    liste de commandes]
`endif`

Sélection cas

`switch` (*valeur*)
[**`case`** *valeur1*:
    liste de commandes
    `breaksw`]
[**`default:`**
    liste de commandes]
`endsw`

2.7 Redirections

$<$ redirige l'entrée standard

$<<$ ***mot*** l'entrée standard est le texte qui suit, jusqu'au prochain *mot* placé seul sur une ligne

$>$ redirige la sortie standard avec création d'un fichier s'il est absent, ou écrasement s'il existe

$> \&$ idem pour sortie standard + erreur standard (en même temps)

$>>$ redirige la sortie standard en allongeant le fichier

$>> \&$ redirige la sortie standard + erreur standard en allongeant le fichier

Pour rediriger à la fois la sortie standard et l'erreur standard sur des fichiers différents, il est nécessaire de créer un processus supplémentaire.

Exemple :

(**`wc`** *présent absent* $>$ *sortie*) $> \&$ *erreur*

2.8 Scripts

Un fichier *f* contenant un texte en langage de commandes constitue un sous-programme en langage de commandes, ou script. Il est activé en C-Shell par la commande **`csh`** *f*, ou, avec les droits *x* sur ce fichier, en exécutant directement *f*.

Pour que le bon interprêteur soit activé à l'exécution de la commande, il est nécessaire de placer en première ligne de *f*

```
#!/bin/csh
```

Chapter 3

Traitements sur les fichiers

3.1 Recherche d’une chaîne dans un fichier

3.2 Tri, fusion

3.3 Collage par concaténation de lignes

3.4 Projection

3.5 Substitution de caractères

3.6 Remplacement des tabulations

3.7 Eclatement d’un fichier

Chapter 4

Sed et awk

4.1 Sed

4.2 Awk