Université Toulouse III Paul sabatier - IUT A
DUT Informatique

Système 1

A.S.R

Première partie

Les commandes de bases

Chapitre 1

Notion de commande

Une commande est une suite de mots séparés par au moins un espace.
Le premier est le nom de la commande, suivi par une liste facultative d'options et d'arguments.

```
commande [-options] [arg1 ... argn]
```

Les majuscules et les minuscules sont differenciées dans le système Unix.

Il est possible d'écrire plusieurs commandes sur la même ligne. Le séparateur de commandes est le " ;"

1.1 Quelques exemples de commandes :

- ls
- ls -l
- ls -l -a
- ls -la
- ls -al
- ls bidule
- ls -l bidule

Remarques : sur la présentation de la syntaxe des commandes, tout texte entre [ ] (crochets) est optionnel.

Chapitre 2

Obtenir de l'aide sous Unix

Il existe des informations en ligne disponibles sur Unix.
Une manière simple d'obtenir ces informations c'est d'utiliser la commande man
Pour obtenir la correspondance des codes ASCII en octal et hexadecimal, tapez la commande : man ascii

Pour avoir la syntaxe de la commande man, tapez : man man

Chapitre 3

La commande *man*

3.1 Définition

Cette commande donne accès au manuel en ligne du système en vue d'obtenir de la documentation.

3.2 Syntaxe

```
man [section] nom de la commande
```

Chapitre 4

Les commandes de gestion de fichiers

Dans ce chapitre, nous allons aborder la notion de système de fichiers, pour cela nous donnerons la définition d'un fichier, les différents types de fichiers existant sous Unix ainsi que les chemins d'accès.
Nous détaillerons aussi les commandes de gestion de fichiers et de catalogues.

Le chapitre se divise en :

- la définition d'un système de fichiers,
- les principales commandes de manipulation de fichiers,
- les principales commandes de manipulation de répertoires.

4.1 Le système de fichier

4.1.1 Qu'est ce qu'un fichier ?

Un fichier UNIX est une suite de caractères non structurée. UNIX n'a pas la notion d'organisation de fichier (indexée, relative, etc ...). A tout fichier est attribué un bloc d'informations appelé noeud d'index ou i-node. Cet i-node contient des informations générales concernant le fichier :

- sa taille (en octets),
- l'adresse des blocs utilisés par le fichier sur le disque,
- l'identification du propriétaire du fichier,
- les droits d'accès des différents groupes d'utilisateurs,
- le type du fichier,
- un compteur de liens,
- les dates des principales opérations (création, mise à jour, consultation).

Remarque : le i-node ne contient pas le nom du fichier.

4.1.2 Les types de fichiers

Dans le système UNIX il existe 3 types de fichiers : Les **fichiers ordinaires** peuvent être :

- des programmes exécutables (compilateurs, éditeurs, tableurs, ...)
- des fichiers texte
- des fichiers de données

Il n'y a pas de format à respecter pour le nom des fichiers UNIX (jusqu'à 256 caractères).
Les **fichiers spéciaux** : Ce sont des fichiers associés à un dispositif d'entrée/sortie (E/S) physique. Ils sont traités par le système comme des fichiers ordinaires mais la lecture et l'écriture sur ces fichiers activent les mécanismes physiques associés (drivers). Il existe 2 types de fichiers spéciaux :

- mode caractére : E/S réalisées caractère par caractère (terminaux, imprimantes, lignes de communication, ...)
- mode bloc : E/S réalisées par blocs de caractères (disques, bandes).

Les **répertoires** : Contiennent les couples (i-node, nom de fichier). On ne peut créer, effacer, lire ou écrire dans des répertoires qu'au moyen de primitives systèmes spécifiques.
Les répertoires sont aussi appelés catalogues ou directories.

4.1.3 Conventions de nommage des répertoires

. (point) désigne le répertoire courant.
.. (point point) désigne le répertoire père du répertoire courant.
/ (slash) désigne la racine de l'arborescence des fichiers.

Dans la désignation d'un chemin, c'est un séparateur de catalogue.
(tilde) désigne le "home directory" de l'utilisateur.

4.1.4 Chemin d'accès à un fichier ou à un répertoire

Le chemin d'accès à un fichier (ou à un catalogue) est la description qui permet d'identifier le fichier (ou le catalogue) dans la hiérarchie du système.
Le chemin d'accès correspond en une suite de noms de répertoires séparés par des caractères / (slash) et terminé par le nom du fichier ou du répertoire.

Ainsi le chemin d'accès suivant : /users/fudmip/prof/.login
représente le fichier .login qui se trouve dans le catalogue profcatalogue de connexion lui-même placé sous le catalogue fudmip, lui-même contenu dans le catalogue users qui se trouve sous la racine /.
Remarque : Le caractère / marque la séparation entre catalogues lorsqu'on décrit le "chemin d'accès" à un fichier ou un catalogue.

4.2 Manipulation de fichiers

4.2.1 Afficher la liste des fichiers d'un répértoire : *ls*

La commande *ls* affiche la liste des fichiers du répertoire courant :

```
$: ls
boite fic fic2 Fich.c
```

4.2.2 Afficher le contenu d'un fichier : *cat*

Définition

Cette commande permet d'afficher le contenu d'un fichier dont le nom est passé en paramètre.
En réalité la commande cat concatène le contenu de tous les fichiers passés en arguments et envoi le résultat sur l'écran

Elle permet aussi de créer un fichier en utilisant la redirection d'E/S (>).

Syntaxe

```
cat essai.c
cat > truc
```

4.2.3 Afficher le contenu du fichier avec arrêt à chaque page : *more*

Définition

Cette commande permet d'afficher le contenu d'un fichier avec arrêt à chaque page. On peut alors utiliser : la touche ESPACE pour passer à la page suivante, la touche h pour avoir l'aide, la touche q ou D̂ pour sortir de more.
Remarque : la commande more fait partie de la famille des pagers , il en existe d'autres pg, less, ...

syntaxe

```
more monFichier
```

4.2.4 Affichage partiel du contenu d'un fichier

Les commandes suivantes permettent d'afficher à l'écran une partie du contenu d'un fichier :
- les premières lignes : *head*
- les dernières lignes : *tail*
- certaines lignes : *grep*
- des parties de lignes : *cut*

Les premières lignes d'un fichier : *head*

Définition Cette commande affiche les premières lignes d'un fichier.
Des options permettent de modifier le nombre de lignes à afficher.

syntaxe

```
head [-n] [fichier1] [fichier2 ...]
```

Les dernières lignes d'un fichier : *tail*

Définition Cette commande affiche les dernières lignes d'un fichier.
Des options permettent d'en modifier le nombre par défaut.

syntaxe

```
tail [+/-n] [fichier]
```

Certaines lignes d'un fichier : *grep*

Définition La commande grep affiche toutes les lignes d'un fichier contenant la chaîne de caractères spécifiée en argument. Il est possible d'utiliser des métacaractères pour définir la chaîne à rechercher.

syntaxe

```
grep  chaîne fichier
```

Une partie des lignes d'un fichier : *cut*

Définition La commande grep affiche toutes les lignes d'un fichier contenant la chaîne de caractères spécifiée en argument. Il est possible d'utiliser des métacaractères pour définir la chaîne à rechercher.

syntaxe

```
grep  chaîne fichier
```

4.2.5 Copier un fichier : *cp*

4.2.6 Définition

Cette commande permet de copier des fichiers.
L'option -R autorise la copie de catalogue.

4.2.7 syntaxe

```
cp   fic-source    fic-cible
cp   fic-source   ktal-cible
cp -R  ktal-source   ktal-cible
```

4.2.8 Copier un fichier : *rm*

Définition

Cette commande permet de détruire les fichiers passés en paramètres.
De nombreuses options de cette commande sont pratiques mais dangereuses !
Par défaut, la commande rm ne demande aucune confirmation : les fichiers sont donc irrémédiablement perdus.

syntaxe

```
rm  -[Rfi] mon-fichier
rm  -i mon-fichier
rm  -f mon-fichier
rm  -R mon-fichier
```

4.2.9 Renommer ou déplacer un fichier : *mv*

Définition

Cette commande permet de déplacer ou de renommer un fichier ou un répertoire.

syntaxe

```
mv mon-fichier nouv-fichier
```

4.2.10 Comparaison de 2 fichiers

UNIX met a disposition 2 commandes pour comparer le contenu de fichiers :
- cmp indique si les contenus des fichiers sont les mêmes.
- diff affiche les modifications à apporter pour les rendre identiques.

4.2.11 Comparaison du contenu de 2 fichiers : *cmp*

Définition

Cette commande permet de comparer le contenu de 2 fichiers. Elle affiche le numéro de ligne et de caractère de la première différence rencontrée.

syntaxe

```
cmp fichier1 fichier2
```

4.2.12 Différence entre 2 fichiers : *diff*

Définition

Cette commande permet d'afficher les différence entre deux fichiers.

syntaxe

```
diff fichier1 fichier2
```

4.2.13 Trier des fichiers : sort

4.2.14 Définition

sort trie, regroupe ou compare toutes les lignes des fichiers indiqués. Si aucun fichier n'est fourni, ou si le nom '-' est mentionné, la lecture se fera depuis l'entrée standard.

4.2.15 syntaxe

```
sort [-cmus] [-t séparateur] [-o fichier_de_sortie] [-T répertoire_temporaire] [-bdfiMnr] [+POS1 [-POS2]]
```

4.2.16 Compter les caractères, mots , lignes d'un fichier : *wc*

4.2.17 Définition

La commande wc compte les mots, les lignes et/ou caractères d'un fichier.

4.2.18 syntaxe

```
wc [-lcw] fichier
```

4.2.19 Numéroter des lignes : *nl*

4.2.20 Définition

La commande nl numéro les lignes d'un fichier et l'affiche à l'écran

4.2.21 syntaxe

```
nl fichier
```

4.3 Manipulation de répertoires

Les répertoires servent à ranger des fichiers et/ou catalogues.

L'espace de travail de l'utilisateur est une arborescence de répertoires et de fichiers.

Des commandes UNIX permettent d'organiser et de gérer cette hiérarchie (créer, effacer, parcourir, ...)

Il y a un catalogue particulier à chaque usager : le catalogue personnel.
Ce catalogue est le sommet de l'arborescence de l'espace de travail de l'utilisateur, c'est le catalogue dans lequel il est placé à la connexion (home directory).

Le catalogue de connexion est repéré par la variable d'environnement HOME, mais aussi par le caractère .
Généralement le nom de votre répertoire personnel est identique à votre nom d'utilisateur.
Les principales commandes sur les répertoires :

- Afficher le répertoire courant : *pwd*
- Se déplacer dans l'arborescence : *cd*
- Créer un répertoire : *mkdir*
- Effacer un répertoire : *rmdir*
- Copier un répertoire : $cp - R$

4.3.1 Afficher le répertoire courant : *pwd*

Le répertoire courant est le catalogue dans lequel vous êtes en train de travailler.
Initialement le répertoire courant est le catalogue de connexion.
Il est nécessaire de connaître sa position dans l'arborescence du système à tout instant.

Définition

La commande pwd affiche à l'écran le chemin d'accès au catalogue courant.

Syntaxe

```
pwd
```

Remarque

Certains systèmes maintiennent une variable PWD qui contient le chemin d'accès au catalogue courant. En shell csh (tcsh, ...) la variable cwd contient aussi le catalogue courant.

4.3.2 Se Déplacer dans un répertoire : *cd*

Définition

Cette commande permet de se déplacer dans l'arborescence des catalogues existants sur le système.

Syntaxe

```
cd
cd [ chemin relatif ]
cd [ chemin absolu ]
```

4.3.3 Créer un répertoire : *mkdir*

Définition

Cette commande permet de créer des répertoires, il faut bien sûr pouvoir le faire, c'est-à-dire être dans son espace de travail.

Syntaxe

```
mkdir [-p] nom-répertoire
```

4.3.4 Détruire un répertoire : *rmdir*

Définition

La commande rmdir permet de détruire des catalogues vides.

Syntaxe

```
rmdir [-f |-i] [-p] répertoire
rm -R   répertoire
```

4.3.5 Copier un répertoire : *cp*

Définition

Il est possible de dupliquer le contenu d'un répertoire en utilisant la commande cp (copy) et l'option -r (récursive). De cette façon, tous les fichiers contenus dans tous les sous-répertoires du répertoire copié seront copiés également.

Syntaxe

```
cp -r répertoire-a-copier nouveau-répertoire
```

Chapitre 5

Notion d'utilisateur

Tout utilisateur est enregistré dans deux fichiers systèmes :
/etc/passwd
Ce fichier contient :
- nom de login
- mot de passe chiffré
- numéro unique d'utilisateur (UID)
- numéro unique de groupe (GID)
- nom complet de l'utilisateur
- répertoire initial
- interpréteur de commande

/etc/group
Ce fichier contient :
- nom de groupe
- numéro unique de groupe (GID)
- liste des utilisateurs du groupe

Remarque : un utilisateur peut faire partie de plusieurs groupes.

Les notions d'utilisateurs et de groupes sont fondamentales pour l'attribution de droits d'accès aux fichiers.

5.1 Protection : Droits d'accès

5.1.1 Permissions et contrôle d'accès

Un ensemble de permissions d'accès est associé à tout fichier ; ces permissions déterminent qui peut accéder au fichier et comment :

Fichier		Répertoire	
r	Accès en lecture	r	Accès en lecture
w	Accès en écriture	w	Accès en création, modification et destruction
x	Accès en exection	x	Accès au nom

Pour accéder à une feuille ou à un noeud dans une arborescence de fichiers, il faut avoir la permission en x sur tous les répertoires de niveau supérieur (chemin d'accès).
Il existe trois classes d'utilisateurs :

u	Propriétaire
g	groupe du propriétaire
o	Les autres

Il existe quatre principaux types de fichiers :

-	Fichier ordinaire
d	Répertoire (directory)
c	Fichier mode caractère
b	Fichier mode bloc

Modification des droits d'accès : *chmod*

Les permissions d'accès peuvent être modifiées par la commande chmod
Seul le propriétaire du fichier (ou répertoire) a la possibilité de modifier les droits d'accès sur son fichier.

syntaxe

```
chmod mode fichier [ou répertoire]
```

Changement de propriétaire : *chown*

Le propriétaire d'un fichier ou d'un répertoire peut être changé par la commande chown
Seul le propriétaire du fichier (ou répertoire) et le SU peuvent effectuer ce changement.

syntaxe

```
chown NouveauPropriétaire fichier [ou répertoire]
```

Changement de groupe : *chgrp*

LeVous pouvez changer le groupe auquel s'applique les protections de niveau groupe par la commande chgrp
Seul le propriétaire du fichier (ou répertoire) et le SU peuvent effectuer ce changement.

syntaxe

```
chgrp NouveauGroupe fichier [ou répertoire]
```

Notion de masque : *umask*

Tout fichier créé par le système d'exploitation a des droits d'accès par défaut.
Par exemple sous HP-VUE un fichier texte a comme droits d'accès 666, un fichier exécutable ou un répertoire a pour droits d'accès 777.
Pour éviter toute intrusion sur ces fichiers le fichier .login ou .profile (selon le Shell utilisé) contient souvent la commande umask 022 qui change les droits d'accès par défaut.

Ces nouveaux droits sont :

- pour un fichier texte, 644,
- pour un fichier exécutable ou un répertoire, 755.
- Pour les répertoires et les fichiers exécutables, les bits à 1 du masque invalident les droits correspondant.
- Inversement, les bits à 0 du masque autorisent les droits correspondant.
- Pour les fichiers texte, même application en enlevant en plus le droit en exécution.

Pour obtenir la valeur courante du masque taper la commande umask
La valeur retournée est donnée en octal.

Bien entendu, la commande chmod ignore les masques

Notion de superutilisateur

Les restrictions d'accès s'appliquent aux utilisateurs.
Un seul utilisateur est exempt de contrôles d'accès :
le Super Utilisateur ==> login : root

Chapitre 6

Notion de procéssus

Un processus est un programme en cours d'exécution.

Un programme, produit par un éditeur de liens, est un fichier binaire exécutable mémorisé sur disque.
Pour l'exécuter le système le charge en mémoire, il devient alors un processus.

Un processus est identifié au sein du système par un numéro unique, le PID.
Les commandes de gestion de processus :

- Etat des processus actifs : *ps*
- Arreter un processus actif : *kill*
- Différentes façons de lancer un processus (*nohup*, *attime*, *batch*, *nice*).

6.1 La commande *ps*

6.1.1 Définition

Cette commande affiche la liste des processus actifs sur le système. Il existe 2 types de processus : les processus systèmes qui accomplissent des services généraux et les processus utilisateurs.
Par défaut, la commande ps n'affiche que les processus utilisateurs.

6.1.2 syntaxe

```
ps [-u utilisateur] [-e] [-f]
```

6.2 La commande *kill*

6.2.1 Définition

Cette commande interrompt un processus en cours d'exécution.
En réalité kill envoie un signal au processus spécifié.

6.2.2 syntaxe

```
kill [-signal] PID
```

6.3 Différentes façon de lancer un processus

6.3.1 Continuité d'exécution d'un processus : *nohup*

Lors de la déconnexion, les processus lancés par l'utilisateur qui s'exécutent encore sont tués automatiquement. Toutefois il est possible de prolonger l'exécution d'un processus en utilisant cette commande.

```
nohup commande [parametres]
```

6.3.2 Exécution d'un processus en différé : *attime*, *batch*

at : permet de spécifier le moment de l'exécution.
batch : la commande est mise en attente et est exécutée quand le système n'est pas surchargé

```
at time commande [parametres]
batch commande [parametres]
```

6.3.3 Exécution d'un processus avec priorité basse : *nice*

La commande nice permet de donner des priorités plus ou moins élevées pour l'exécution de processus selon l'importance de la tâche qu'ils remplissent.

```
nice [+/-nombre] commande [parametres]

avec 1 (fort) <= nombre <= 19 (faible).
Par défaut, nombre est égal à 10.
```

Remarque : Seul le SU (Super Utilisateur) peut exécuter un processus en augmentant sa priorité. Un utilisateur normal ne peut que la diminuer.

Deuxième partie

Le langage Shell

Ce chapitre traite du shell originel d'UNIX : le Shell de Bourne.
Nous verrons en détail les spécificités de cet interpréteur de commandes, et nous aborderons la programmation de shell script de Bourne.

Chapitre 7

Sh : Le shell de bourne

7.1 Introduction

Le Shell Unix est non seulement un interpréteur de commandes mais aussi un langage de programmation.
Il interprète et exécute les commandes UNIX.
Cependant, l'utilisateur peut construire, à l'aide de structures de contrôle du Shell, et mémoriser ses propres ensembles de commandes dans des fichiers de commandes appelés script Shell .

- portabilité
- facilité de mise en oeuvre
- simplicité

7.2 Les facilités du shell

- génération des noms de fichiers
- variables non typées
- paramètres positionnels
- substitution de commandes ... etc
- commandes cataloguées
- caractères spéciaux et autres notations

7.3 Les variables

7.3.1 Définition

La notion de variable existe aussi en UNIX. Mais contrairement à un langage déclaratif tel que Pascal, l'utilisateur n'a pas besoin de les déclarer.
Les identificateurs sont composés de caractères alphanumériques mais le premier doit obligatoirement être alphabétique.
Exemple :

```
position
d1
A320
```

Les variables sont de type chaîne de caractères.
Pour obtenir le contenu d'une variable, il faut la faire précéder de $.
Deux types de variables :

- **locales**, créées par l'utilisateur (usage local à un processus)
- d'**environnement**, créées par le système ou par l'utilisateur

7.3.2 Les variables Shell ou locales

création et initialisation :
$nom = valeur$
la valeur est une chaîne de caractères

exemple : $cours = unix$
===> chaîne 'unix'
exemple : $poids = 156$
===> chaîne '1' '5' '6'
La valeur de la variable poids n'est pas le nombre 156.

interprétation numérique d'une chaîne de chiffres
commande *test*
commande *expr*

remarques :
chaîne vide : " ou "" ou rien ===> x=" ou x="" ou x=
'-' (souligné) illégal dans un nom de variable
'-' ok pour une valeur (signe négatif)

Pour accèder à la valeur d'une variable on utilise la commande echo
L'écho d'une variable non définie ou d'une variable affectée d'une chaîne vide produit une ligne blanche.

7.3.3 Les variables Shell ou locales

Elles sont nécessaires au fonctionnement du shell, du système voire de certaines applications.

La commande set imprime toutes les variables du shell
La commande env liste les variables d'environnement
Certaines de ces variables sont modifiées dans le .profile ou dans le fichier nommé dans la variable d'environnement ENV (ksh)

7.3.4 Variables shell locales et d'environnement

- Une variable ordinaire est une variable locale ; elle n'est connue que dans le shell où elle a été créée.
- Une variable de même nom créée dans un sous-shell, sera une variable distincte. La valeur de la même variable dans le shell père n'est pas modifiée.
- Tout shell a ses propres variables locales.
- Tout shell a ses propres variables "globales" appartenant à son environnement ; elles sont héritées par les sous-shells.

7.3.5 Transmission de variables

commande *export*
Cette commande permet de placer une variable dans l'environnement courant. Toute variable utilisée en argument de cette commande possèdera une copie dans tous les sous-shell

Remarque : les variables peuvent être exportées vers les sous-shell, mais ne peuvent pas être renvoyées vers le shell initial.

7.4 Shell scripts

Les shells scripts interactifs sont basés sur des commandes shell (read et echo) ainsi que sur l'usage de variables.

Remarques :
- on peut lire une liste de variables
- on peut répondre en tapant plusieurs mots sur la même ligne

7.4.1 Arguments

La plupart des commandes shell acceptent des arguments :

Même approche pour les shell scripts les arguments sont référencés par leur position dans la liste des arguments :
- $1 premier argument,
- $2 second argument... etc

  conventions :
- $0 : nom de la commande elle-même
- $* : liste des arguments
- $: nombre d'arguments

7.4.2 Substitution de commande

méta-caractère antiquote ‘ (backquote)

Lorsque une chaîne de caractères est entourée par ‘, elle est interprétée comme une commande et remplacée par son résultat

```
$: echo date
date

$: echo voici la date ‘date‘
voici la date Vendredi 22 mars 1996 12:23:00
```

7.4.3 Valeur résultat de l'exécution d'une commande

- Exécution correcte ==> 0
- Exécution incorrecte ==> != 0
- Exemple : grep mot fichier
- Valeur retournée par la commande :
- 0 : 1 ou plusieurs mots mot trouvés
- 1 : mot non trouvé
- 2 : syntaxe incorrecte

7.4.4 Variables spéciales

- $: nombre de paramètres positionnels
- $ - : options du shell
- $? : valeur résultat (état de sortie) de la dernière commande exécutée
- $ $: numéro de processus du shell courant
- $! : numéro de processus du dernier processus en background
- $ 0 : nom de la commande en cours
- $ * : liste des paramètres positionnels

7.5 Programmation Shell de bourne

Le shell de Bourne offre les structures de contrôle d'exécution de commandes d'un langage de programmation traditionnel.

Conditions et Choix
if/then/else
Expression d'une condition : *test*
case

Répétitions *for*
while
until
Sortie impérative : *break*

7.5.1 Conditions

Définition

La structure de contrôle if/then/else permet de choisir l'exécution d'un ensemble de commandes suivant une condition.

Syntaxe

La syntaxe if/then/else peut-être plus ou moins complète, comme nous la détaillons dans la syntaxe. il est possible d'imbriquer plusieurs if.

```
if condition
    then
        instruction
        ...
        fi
```

else :

```
if condition
    then
        instruction
        ...
    else
        instruction
        ...
    fi
```

elif :

```
if condition
    then
           instruction
        ...
    elif condition
        then
            instruction
            ...
           fi
```

7.5.2 test

Définition

La commande *test* permet d'effectuer des tests d'existence et de comparaison.
Les différents types de test sont :
- Test d'état de ficiher
- Test de comparaison de chaînes
- Test de comparaison numérique

Syntaxe

Test d'état de fichier

$test - rfichier$	Vrai, si existe et droit en lecture
$test - wfichier$	Vrai, si existe et droit en écriture
$test - ffichier$	Vrai, si existe et non répertoire
$test - dfichier$	Vrai, si existe et repertoire
$test - sfichier$	Vrai, si existe et taille > 0
$test - xfichier$	Vrai, si existe et exécutable

Test de comparaison de chaînes

$testchane1 = chane2$	Égalité
$testchane1! = chane2$	Non égalité
$test - nchane$	longueur non égal à 0
$test - zchane$	longueur zéro

Test comparaison numérique

$testn1 - eqn2$	Égalité
$testn1 - nen2$	Égalité
$testn1 - gtn2$	Égalité
$testn1 - ltn2$	Égalité
$testn1 - gen2$	Égalité
$testn1 - len2$	Égalité

Les opérateurs logiques sont aussi utilisés avec la commande *test*

!	Négation
$-a$	Et logique
$-o$	Ou logique

7.5.3 For

Définition

La structure de contrôle de répétion for permet d'exécuter les commande ssituées entre le *do* et le *done* pour chacune des valeurs prise dans la liste fournie après le *in*

Syntaxe

```
for variable in liste-de-valeurs
    do
        commandes
    done
```

7.5.4 While

Définition

La structure de contrôle de répétion *while* permet d'exécuter les commande ssituées entre le *do* et le *done* **tant que** la condition est vraie.

Syntaxe

```
while condition
    do
        liste-de-commandes
    done
```

7.5.5 Until

Définition

La structure de contrôle de répétion *until* permet d'exécuter les commande ssituées entre le *do* et le *done* **jusqu'à ce que** la condition soit vraie.
Notez que contrairement à certains langages de programmation la condition est évaluée avant de rentrer dans la boucle.

Syntaxe

```
until condition
    do
        liste-de-commandes
    done
```

7.5.6 Quelques commandes utiles pour programmer

Break

L'instruction *break* permet de sortir des structures de répétition *while*, for, et *until*

expr

La commande *expr* permet d'évaluer une expression arithmétique ou d'effectuer des opérations sur les chaînes de caractères

set

La commande *set* permet de mettre en oeuvre certaines options :
$set - x$ permet d'afficher les commandes à exécuter

shift

L'insctruction $shift$ permet de décaler vers la gauche la liste des paramètres passés en argument à un scription. La valeur de $n+1 est mise dans $n

exit

L'insctruction *exit* autorise la sortie impérative d'un script en cours d'exécution.