

# Introduction à UNIX

Licence d'informatique

Vincent Ribaud et Stéphane Rubini  
Université de Bretagne Occidentale

21 septembre 2001



# Table des matières

<b>1</b>	<b>Principes d'UNIX</b>	<b>5</b>
1.1	Introduction . . . . .	5
1.2	Une première décomposition d'un système informatique . . . . .	5
1.2.1	Définition d'un système informatique . . . . .	5
1.2.2	Architecture de la plate-forme X/OPEN . . . . .	6
1.3	Premiers pas en UNIX . . . . .	7
1.3.1	Compte et mot de passe . . . . .	7
1.3.2	Procédure de connexion . . . . .	7
1.4	Les commandes . . . . .	9
1.4.1	L'interpréteur de commandes . . . . .	9
1.4.2	Entrée des commandes . . . . .	9
1.4.3	Caractéristiques et fichiers de démarrage . . . . .	10
1.4.4	Historique . . . . .	11
1.4.5	Les caractéristiques du terminal . . . . .	11
1.5	La documentation . . . . .	11
1.6	Le système de fichier . . . . .	12
1.6.1	Principes . . . . .	12
1.6.2	L'arborescence des fichiers . . . . .	13
1.6.3	Noms des fichiers . . . . .	13
1.6.4	Renommer et détruire des fichiers . . . . .	16
1.6.5	Droits d'accès aux fichiers . . . . .	16
1.6.6	Les fichiers spéciaux . . . . .	17
1.7	Retour sur l'interpréteur de commandes . . . . .	17
1.7.1	Expansion des noms de fichiers . . . . .	18
1.7.2	Entrée et sortie standard . . . . .	18
1.7.3	Redirection . . . . .	18
1.7.4	Les tubes et les filtres . . . . .	19
1.7.5	Variables d'environnement . . . . .	20
1.7.6	Les alias . . . . .	20
1.7.7	La substitution de commande . . . . .	21
1.7.8	Les fichiers de commandes . . . . .	21
1.7.9	Commandes diverses . . . . .	21

<b>2</b>	<b>L'éditeur de texte vi</b>	<b>23</b>
2.1	Passage en mode insertion . . . . .	24
2.2	Le mode <i>commande</i> . . . . .	24
2.3	Le mode <i>ex</i> . . . . .	24
<b>3</b>	<b>L'éditeur de texte ex</b>	<b>25</b>
3.1	Les commandes <i>ex</i> . . . . .	25
3.1.1	Symboles d'adressage des lignes . . . . .	25
3.1.2	Les commandes . . . . .	26
3.2	Les expressions régulières . . . . .	26
	<b>Bibliographie</b>	<b>26</b>

# Chapitre 1

## Principes d'UNIX

### 1.1 Introduction

Un *système d'exploitation* est un programme qui gère l'ensemble du matériel et du logiciel mis à la disposition d'utilisateurs.

UNIX est un système d'exploitation multi-utilisateurs. Il permet de partager une machine entre plusieurs personnes, et d'exécuter plusieurs traitements simultanément.

Les fonctions principales des systèmes d'exploitation (et donc d'UNIX) sont :

- la gestion des ressources matérielles permet de partager les ressources entre les différentes tâches et utilisateurs de façon transparente pour ces derniers,
- la gestion du réseau permet de partager les ressources entre les différentes machines interconnectées,
- la gestion du système de fichier permet de nommer et d'utiliser de manière uniforme les entités externes : fichiers, répertoires, entrées-sorties,
- la gestion des processus permet d'effectuer tous les travaux utilisateurs et systèmes,
- les interpréteurs de commande ou *shells* permettent de créer de nouveaux utilitaires systèmes, de combiner des travaux entre eux, de rediriger les entrées-sorties.

### 1.2 Une première décomposition d'un système informatique

#### 1.2.1 Définition d'un système informatique

Un système informatique est un assemblage de composants matériels (processeur, mémoire, ...) et de composants logiciels (système d'exploitation, base de données, système de fenêtrage, ...) souvent fabriqués et distribués par des sociétés différentes.

Le terme *système* est indifféremment employé pour désigner un système informatique, ou système d'exploitation.

Le terme *architecture d'un système* désigne l'ensemble des principes et techniques mis en œuvre pour concevoir et réaliser cet assemblage de composants.

L'architecture d'un système informatique est structurée en *couches*, chaque couche est construite à partir de la couche immédiatement inférieure, et masque celle-ci pour la couche immédiatement supérieure.

Autrement dit, la couche  $i$  s'appuie sur les fonctions proposées par la couche  $i-1$ , et fournit des services à la couche  $i+1$ .

L'architecture en couche la plus triviale distingue deux couches :

- la couche matérielle ou *hardware*<sup>1</sup>,
- la couche logicielle ou *software*<sup>2</sup>.

Avec le développement des réseaux informatiques, la littérature spécialisée a introduit une couche "fourre-tout" entre matériel et logiciel : le *middleware*, qui regroupe un certain nombre de services utilisées par les machines en réseau.

### 1.2.2 Architecture de la plate-forme X/OPEN

X/Open est un consortium international de fournisseurs de matériels informatiques et de systèmes d'exploitation. Les membres d'X/Open sont des concurrents, mais la pression du marché contraint les fabricants à proposer des systèmes dits *ouverts*, capables de coopérer. X/Open est un moyen pour les constructeurs de trouver un accord sur la coopération entre leurs différents systèmes informatiques.

C'est aussi le but des organismes de normalisation internationaux comme l'ISO<sup>3</sup> ou nationaux (AFNOR en France), mais la normalisation est un processus compliqué et relativement long, qui ne convient pas toujours aux impératifs économiques.

Dans le jargon des systèmes ouverts, une couche s'appuyant sur un regroupement de couches imbriquées constitue une *plate-forme* pour la couche immédiatement supérieure. Une plate-forme offre ainsi une base de services sur laquelle de nouvelles couches peuvent venir s'appuyer ou être développées.

La description et le moyen d'accès à ces services sont appelés l'interface de la plate-forme, le terme interface représentant la partie "en contact" entre deux couches :

- la couche inférieure qui fournit les services,
- la couche supérieure qui utilise les services.

Seules les spécifications fonctionnelles (la description des services fournis) et les spécifications techniques (la manière d'invoquer ces services, généralement par le biais d'un langage de programmation donné) sont publiées.

L'ensemble des plate-formes adoptées par X/Open porte le nom de *plate-forme X/Open*, son architecture est donnée à la figure 1.2.2.

---

1. littéralement : quincaillerie  
 2. par opposition à HARDware  
 3. International Standardization Organisation

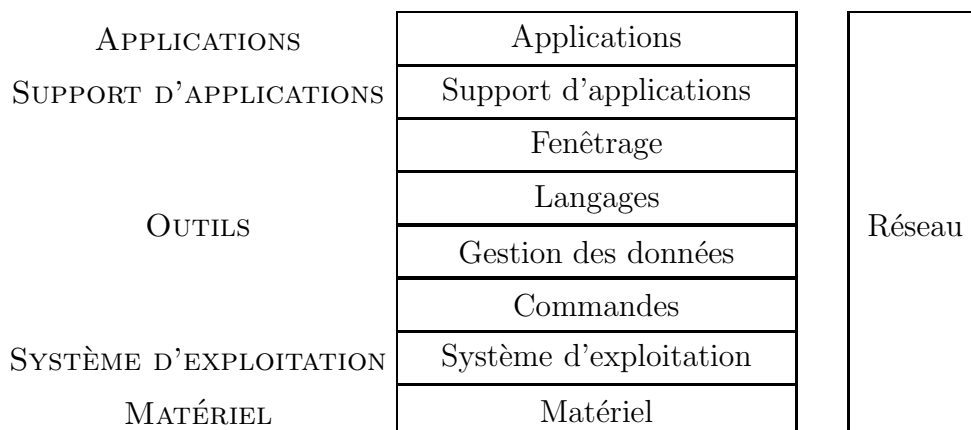


FIG. 1.1 – Architecture de la plate-forme X/Open

L'interface de la plate-forme Système d'exploitation, adoptée par X/Open est la norme internationale IEEE P1003.1, plus connue sous le nom de POSIX.

Bien que développée à partir de l'interface SVID de l'UNIX d'AT&T, POSIX est une interface de système d'exploitation ouvert et peut être mise en oeuvre au-dessus d'un autre système d'exploitation qu'une variante d'UNIX.

La plupart des développeurs s'accordent à reconnaître que cette interface POSIX est indépendante des constructeurs et qu'elle est suffisamment générale pour pouvoir être mise en oeuvre sur un grand nombre de systèmes d'exploitation.

## 1.3 Premiers pas en UNIX ...

### 1.3.1 Compte et mot de passe

Avant que vous puissiez vous connecter, vous devez avoir un compte. Le nom de votre compte est formé en général des huit premiers caractères de votre nom, prefixé de (des) initiales(s) de votre prénom, s'il existe déjà un utilisateur portant le même nom que vous.

De plus, on doit vous donner un mot de passe, qui ne doit être connu que de vous, et sera entré lors de la procédure de connexion, pour s'assurer que vous êtes bien l'utilisateur dont vous vous réclamez.

### 1.3.2 Procédure de connexion

Quand vous êtes prêts à vous connecter, vous devez avoir sur l'écran quelque chose qui ressemble à ça :

```
login:
password:
```

Après que vous ayez entré votre nom d'utilisateur et votre mot de passe, la machine vérifie votre mot de passe, et si tout est correct, lance votre environnement de travail.

Ce qui se passe réellement dépend de votre configuration initiale, et du type du terminal qui est devant vous. Si vous utilisez une station de travail ou un terminal X, plusieurs fenêtres apparaissent généralement.

Après la connexion, l'ordinateur affiche l'*invite* ou "prompt". L'invite est une suite de caractères qui indique que le système d'exploitation (en réalité un outil appelé *interpréteur de commande* ou "shell") est prêt à travailler pour vous.

Une fois que vous êtes connecté à une machine reliée au réseau, vous pouvez aussi vous connecter aux autres machines sur lesquelles vous avez un compte. Il faut cette fois utiliser le programme `rlogin` (`r` pour *remote*, à distance) ou `telnet`. Contrairement à la connexion initiale, il faut donner le nom de la machine sur laquelle on se connecte. Par exemple :

```
$ rlogin machine.toto.ailleurs
```

Si, sur l'autre machine, votre nom d'utilisateur est différent, on utilisera :

```
$ rlogin -l mon_autre_nom machine.toto.ailleurs
```

Une session telnet ressemble à ça :

```
telnet machine.toto.ailleurs
Trying 7.7.7.7 ...
Connected to machine.
Escape character is '^]'.

BIDULE OS UNIX (machine)

login: nom_utilisateur
Password:
```

Pour se déconnecter, vous pouvez taper la commande `logout` ou `exit` ou taper `^d`, ou taper `^]`, qui signifie "maintenir la touche Ctrl enfoncée, et, pendant ce temps, appuyer sur la touche d".

A la première connexion, changez votre mot de passe. Le changement de mot de passe s'effectue avec la commande `passwd`.

```
$ passwd
Enter login(NIS) password:
New password:
Re-enter new password:
NIS(YP) passwd/attributes changed on on machine_NIS
```

## 1.4 Les commandes

### 1.4.1 L'interpréteur de commandes

Un interpréteur de commandes (vraisemblablement de type *cshell*) est un programme qui sert d'intermédiaire entre l'utilisateur (une personne physique) et le système (un ensemble de programmes).

Le service essentiel rendu par un interpréteur de commandes est le lancement de programmes exécutables. En simplifiant, un interpréteur de commandes exécute une boucle infinie sur les actions suivantes:

- lecture d'une ligne,
- interprétation de cette ligne comme une demande d'exécution d'un programme avec des paramètres,
- lancement de ce programme avec passage des paramètres.

### 1.4.2 Entrée des commandes

Dans chaque fenêtre ou *terminal virtuel*, l'utilisateur est en relation avec un interpréteur de commandes (mettons un *cshell*). Les commandes sont des suites de mots séparées par des blancs. Le premier de ces mots spécifie le nom du programme à exécuter, les autres constituent des paramètres qui sont passés par l'interpréteur de commandes à ce programme :

Il est important de comprendre que ce qu'on appelle une **commande** est un programme exécutable, soit directement par la machine (dans un format adéquat), soit par un interpréteur de commandes (dans un format texte).

Il est donc tout à fait naturel sous UNIX qu'un utilisateur enrichisse les commandes avec ses propres commandes.

Exemple : Quelle est la date, l'heure ?

```
$  
$ date  
Sat Sep 20 10:55:05 MET DST 1997  
$
```

Autre exemple : Qui suis-je ?

```
$ who is moi  
moi pts/4 Sep 18 09:15 (ma_machine)  
moi pts/0 Sep 20 09:16 (ma_machine)  
moi pts/1 Sep 20 09:38 (ma_machine)  
$
```

Tous les autres services offerts par l'interpréteur de commandes n'ont d'autre but que de faciliter cette tâche fondamentale qu'est le lancement de programme exécutable.

Pour exécuter une commande, l'interpréteur de commandes crée un nouveau processus et attend qu'il soit terminé avant d'interpréter la commande suivante.

On peut toutefois lancer une commande sans avoir à attendre sa fin, en ajoutant `&` à la fin. On parle d'exécution en *arrière-plan*.

### 1.4.3 Caractéristiques et fichiers de démarrage

Lorsque UNIX lance un interpréteur de commandes, ses caractéristiques sont *configurables*, c'est à dire qu'on peut définir l'invite ("prompt"), dire à la machine où rechercher les commandes, etc ... Pour faire cela, l'interpréteur de commande exécute au moins un fichier de démarrage. La plupart en exécutent deux (voir table).

L'interpréteur de commandes peut aussi avoir un mécanisme d'historique qui permet de rappeler les dernières commandes, et permet de les répéter et de les modifier.

Une autre caractéristique importante est le contrôle des travaux, qui vous permet d'en arrêter ou d'en lancer en tâche de fond, de telle façon que le terminal n'attende pas que ces travaux soient terminés, et vous redonne immédiatement la main. Vous pouvez alors continuer à entrer des commandes pendant que le travail s'effectue en arrière-plan.

La figure 1.4.3 donne les caractéristiques des différents interpréteurs de commandes.

<i>shell</i>	sh	csH	tcsh	ksh	bash
<i>1<sup>er</sup> fichier de démarrage</i>	<code>.profile</code>	<code>.cshrc</code>	<code>.cshrc</code>	<code>.profile</code>	<code>.bash_profile</code> ou <code>.profile</code>
<i>2<sup>ème</sup> fichier de démarrage</i>	-	<code>.login</code> uniquement pour le shell de login	<code>.login</code> uniquement pour le shell de login	<code>\$ENV</code>	<code>\$ENV</code>
<i>Caractéristiques</i>	simple et portable	contrôle de tâches  historique simple	contrôle de tâches  historique évolué	contrôle de tâches  historique évolué	contrôle de tâches  historique évolué

FIG. 1.2 – *Caractéristiques des principaux interpréteur de commandes.*

sh est aussi appelé Bourne Shell.

`$ENV` dans la ligne "deuxième fichier de démarrage" signifie que le nom du deuxième fichier de démarrage est déterminé par la variable `ENV` dans le premier fichier de démarrage.

Typiquement, `.kshrc` est utilisé pour ksh, et `.bashrc` pour bash. Le signe `$` utilisé devant le nom d'une variable de l'interpréteur de commandes signifie que l'on va insérer à cette place le contenu de cette variable.

### 1.4.4 Historique

Les désignateurs suivants permettent de sélectionner un événement (une ligne) dans la liste historique (en `csh` et `bash`).

<i>opération</i>	<i>commande</i>
<i>Montrer les dernières commandes</i>	<code>history</code>
<i>Répétition de la dernière commande</i>	<code>!!</code>
<i>Répétition de la n<sup>ième</sup> commande</i>	<code>!n</code>
<i>Répétition de la dernière commande commençant par exp</i>	<code>!exp</code>
<i>Répétition de la dernière commande en remplaçant a par b</i>	<code>^a^b</code>

### 1.4.5 Les caractéristiques du terminal

La commande `stty` permet de modifier la fonction associée à une touche du clavier. Par exemple, `stty erase ~` affecte la fonction d'effacement d'un caractère au caractère `~`.

Les caractères de contrôle sont très utilisés pour le contrôle du terminal. Par exemple, l'appui simultané de la touche `Control` et de la touche `c` arrête le programme en cours d'exécution.

La convention de représentation d'un tel caractère peut être :

- `Control-c`
- `^ c`
- `CTRL-c`

## 1.5 La documentation

UNIX offre plusieurs outils d'aide en ligne :

- `apropos expression` est adapté si vous ne vous rappelez pas d'une commande, ou que vous ne vous souvenez pas le nom d'une commande dont vous connaissez les caractéristiques. Le système va chercher `expression` dans les en-têtes des pages des manuels, et vous dira dans quel contexte apparaît `expression`.
- `whatis commande` donne une brève description de `commande`.
- `man commande` vous donne toute l'information sur `commande`. Cette commande rappelle les pages de manuel.

Il peut arriver que `apropos` et `whatis` ne fonctionnent pas sur votre machine, dans ce cas, il ne vous reste que le `man` ...

Voici un exemple de `man` :

```
ECHO(1)                USER COMMANDS                ECHO(1)
```

#### NAME

`echo` - echo arguments to the standard output

#### SYNOPSIS

```
echo [ -n ] [ argument ... ]
```

#### DESCRIPTION

`echo` writes its arguments on the standard output. Arguments must be separated by SPACE characters or TAB characters, and terminated by a NEWLINE.

`echo` is useful for producing diagnostics in shell programs and for writing constant data on pipes. If you are using the Bourne shell (`sh(1)`), you can send diagnostics to the standard error file by typing:

```
echo ... >&2
```

#### OPTIONS

`-n` Do not add the NEWLINE to the output.

#### SEE ALSO

`sh(1)`

Sun Release 4.1 Last change: 9 September 1987

1

## 1.6 Le système de fichier

### 1.6.1 Principes

UNIX voit un *fichier* comme un flux ininterrompu d'informations. On peut aussi le voir comme une suite d'octets. Un octet est une petite quantité d'information stockée sur un ordinateur, qui peut servir à représenter un caractère ou un nombre.

La notion de fichier permet à un processus de se référer à un ensemble logiquement contigu d'octets en utilisant un nom symbolique (le nom du fichier) et des opérations prédéfinies (lire/modifier le contenu, chercher un mot, ...)

Le *système de fichier* est une abstraction qui permet d'avoir un modèle d'entrée-sortie *uniforme* quelque soit le périphérique utilisé (disque, disquette, clavier, imprimante, ...).

Un type particulier de fichier, qui contient les noms des autres fichiers est appelé un répertoire (directory). Les répertoires sont utilisés pour organiser les fichiers présents sur la machine.

Comme nous le verrons plus loin, les canaux d'entrée et de sortie sont aussi des fichiers, mais d'un type spécial.

### 1.6.2 L'arborescence des fichiers

La structure des fichiers sous UNIX est hiérarchique. Elle commence par un répertoire racine, /, et se développe en arborescence.

La figure 5 montre une petite partie de cet arbre, Avec un fragment du répertoire-utilisateur (*home directory*) de l'utilisateur "arthur".

Le *répertoire-utilisateur* (ou `home directory`) d'un utilisateur est le répertoire où l'utilisateur est placé au moment de sa connexion. Ici, `/home/projet2/arthur` est le répertoire-utilisateur de "arthur". Sur la plupart des systèmes, le répertoire-utilisateur d'un individu peut être obtenu par `~nom_d'utilisateur` en `csh`, `ksh` et `bash`.

Votre répertoire personnel peut être obtenu par le seul `~`.

Il n'y a pas de limite au nombre de sous-répertoires. C'est une bonne façon de procéder que de structurer votre répertoire, et créer des sous-répertoires par thèmes. Un répertoire appelé `bin` sert généralement à ranger les fichiers exécutables ("binaires").

Pour gérer les répertoires, on peut utiliser les commandes suivantes :

`pwd` pour afficher le nom du répertoire dans lequel on se trouve (répertoire de travail).

`mkdir nom_du_repertoire` pour créer un répertoire s'appelant `nom_du_repertoire`.

`rmdir nom_du_repertoire` pour détruire le répertoire `nom_du_repertoire`, à condition qu'il soit vide.

`cd nom_du_repertoire` pour aller dans le répertoire `nom_du_repertoire`.

`cd` pour aller dans le répertoire-utilisateur.

`ls` pour afficher le contenu du répertoire.

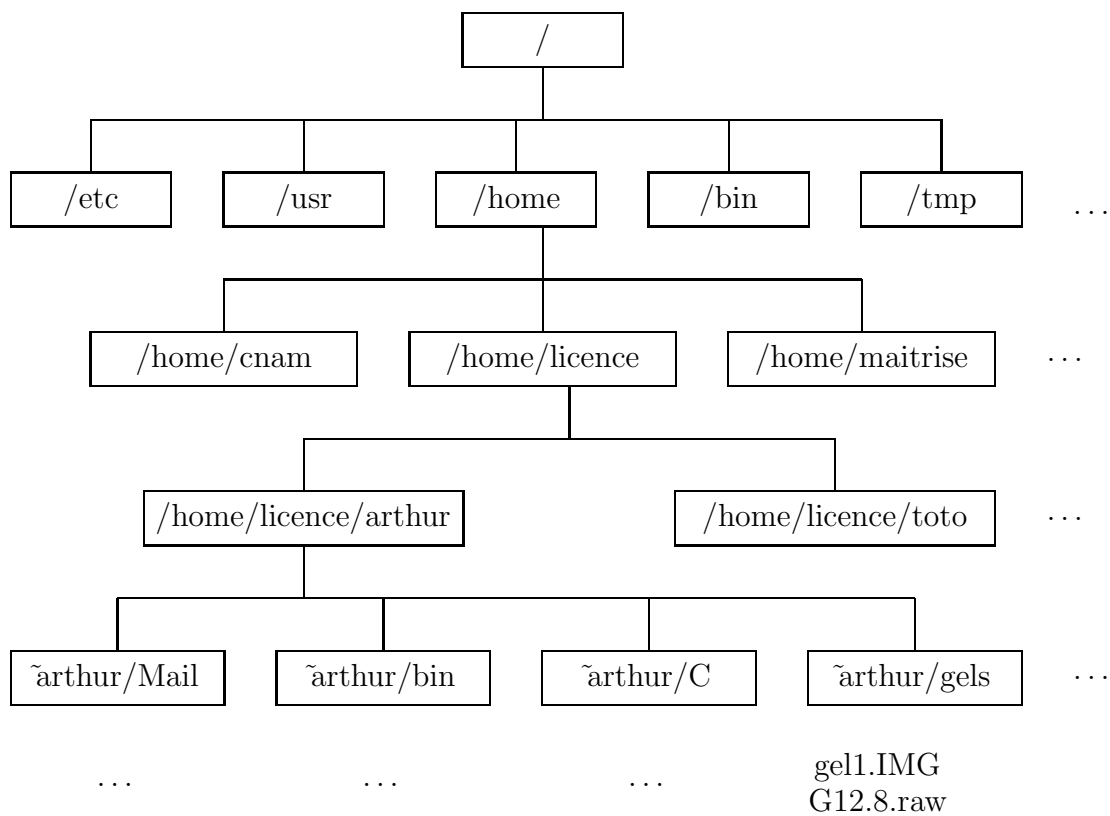
`ls -l` pour afficher le contenu du répertoire, avec plus d'informations sur le contenu.

Le *répertoire courant* (celui où on se trouve, appelé aussi *répertoire de travail* ou `working directory`) est représenté par un point (`.`), le répertoire dont il est issu par deux points (`..`).

### 1.6.3 Noms des fichiers

UNIX n'est pas trop exigeant sur les noms des fichiers, et accepte en principe tous les caractères. Toutefois, il est plus sûr de n'utiliser que les caractères suivants :

- lettres majuscules et minuscules (non accentuées)

FIG. 1.3 – *Éléments d'une structure standard de répertoires sous UNIX*

- chiffres
- blanc souligné (\_)
- point (.)
- tiret (-), sauf en début de nom

Les autres caractères pourraient être interprétés de façon erronée par l'interpréteur de commandes. Si vous utilisez des majuscules, gardez à l'esprit que UNIX différencie majuscules et minuscules. Les noms de fichiers peuvent avoir jusqu'à 253 caractères de long.

Contrairement à MS-DOS, UNIX ne gère pas d'extensions (quelque chose séparé du reste du nom du fichier par un point). En conséquence de quoi, vous pouvez utiliser autant de points qu'il vous plaira dans vos noms de fichiers. Par exemple, UNIX n'a pas besoin d'une terminaison particulière pour décider si un fichier est exécutable ou non. Certaines applications, toutefois, regardent les derniers caractères du nom du fichier séparés par un point pour déterminer de quelles sorte de fichier il s'agit. Parmi ces extensions qui ont une signification particulière pour les programmes, on a :

**.c** code source en langage C

**.o** code objet

**.ps** fichier PostScript (PostScript est un langage graphique).

**.tar** fichier archive

**.Z** fichier compressé

Si vous voulez accéder à un fichier, vous pouvez utiliser son chemin relatif ou absolu.

Le chemin absolu contient toutes les étapes à partir du répertoire racine (/), jusqu'au répertoire dans lequel est placé le fichier.

`/home/projet2/arthur/gels/gel1.IMG` est un exemple de chemin absolu.

Un chemin relatif détermine la voie à partir du répertoire courant. Les chemins relatifs suivants décrivent le même fichier :

<i>Chemin relatif</i>	<i>Répertoire courant</i>
<code>gel1.IMG</code>	<code>/home/licence/arthur/gels</code>
<code>gels/gel1.IMG</code>	<code>/home/licence/arthur</code>
<code>../gels/gel1.IMG</code>	<code>/home/licence/arthur/C</code>

Lorsqu'on se réfère au simple nom de fichier, indépendamment du chemin, on emploie le terme de "base filename".

### 1.6.4 Renommer et détruire des fichiers

`mv fich1 fich2` pour renommer le fichier `fich1` en `fich2`. Si `fich2` existait déjà, il est détruit.

`mv fich rep` pour déplacer le fichier `fich` dans le répertoire `rep`. Le nom du fichier ne change pas, seul son chemin change. Si le fichier `fich` existait déjà dans `rep`, il est détruit.

`mv rep1 rep2` pour déplacer tous les fichiers de `rep1`, y compris les sous-répertoires, dans le répertoire `rep2`. Si `rep2` existe, `rep1` en devient un sous-répertoire. Si `rep2` n'existe pas, il est créé.

`cp fich1 fich2` pour copier le fichier `fich1` en un fichier `fich2`. Si `fich2` existait déjà, il est détruit.

`cp fich rep` copie `fich` dans `rep`, en gardant le nom du fichier.

`cp -r rep1 rep2` pour copier récursivement tous les fichiers et les sous-répertoires de `rep1` dans `rep2`, en préservant l'arborescence.

`rm fich` pour détruire le fichier `fich`. Contrairement à MS-DOS, cette commande est irréversible !

### 1.6.5 Droits d'accès aux fichiers

UNIX permet un contrôle d'accès en répartissant les utilisateurs en trois ensembles :

- le propriétaire d'un fichier,
- d'autres utilisateurs appartenant un ensemble bien défini, appelé *groupe*,
- le reste des utilisateurs

Par exemple, chaque utilisateur étudiant en licence peut être défini comme appartenant au groupe `licence`, les étudiants en maîtrise appartenant au groupe `maitrise`.

En faisant un `ls -l gel1.*` dans le répertoire `~arthur/gels`, on a la sortie suivante :

```
-rwxr-xr--  1 arthur      48436 Oct 12 11:01 gel1.IMG
```

Le premier tiret signifie que vous avez un simple fichier. Dans le cas d'un répertoire, vous verriez ici un "d". Les 9 caractères suivants représentent les droits d'accès au fichier. Le nombre 1 est un compteur de liens<sup>4</sup>. Viennent ensuite le nom du propriétaire, la taille en octets, et la date de dernière modification. En dernier, on trouve le nom du fichier.

---

4. Ceci vous dit s'il y a des liens, c'est à dire d'autres noms pour ce fichier.

Il y a trois choses majeures qu'on peut faire avec un fichier : le lire, l'écrire, et l'exécuter. Les droits d'accès déterminent qui est autorisé à faire chacune de ces trois choses. Les permissions sont établies pour les trois sortes d'utilisateurs : le propriétaire (u), le groupe (g), et le reste du monde (o).

Les trois premiers caractères de la zone droits d'accès vous apprennent que le propriétaire peut lire (r), écrire (w) et exécuter (x) ce fichier. Les trois caractères suivants donnent les permissions pour le groupe. Les membres du groupe peuvent lire et exécuter le fichier, mais pas écrire dessus. Pour voir le nom du groupe s'afficher lors du `ls`, utiliser l'option `-lg`. Les trois derniers caractères indiquent que le fichier peut être lu, mais non réécrit ni exécuté par les autres utilisateurs qui n'appartiennent pas au groupe.

Les droits d'accès peuvent être modifiés par la commande `chmod`, *uniquement par le propriétaire du fichier*. Ceci peut être fait pour le propriétaire (u), le groupe (g), le s autres (o), et l'ensemble des trois (a). Un signe + ou - indique soit l'ajout, soit le retrait de la permission considérée, qui est spécifiée par r, w, ou x. Un signe = assigne explicitement la permission.

```
chmod u-x gel1.IMG
```

rend le fichier `gel1.IMG` non exécutable pour arthur.

```
$ ls -l gel1.IMG
-rw-r-xr-- 1 arthur      48436 Oct 12 11:01 gel1.IMG
```

## 1.6.6 Les fichiers spéciaux

A chaque périphérique d'entrée-sortie correspond un fichier spécial. Ils sont habituellement rangés dans le répertoire `/dev`.

Les fichiers spéciaux utilisent les mêmes interfaces que les fichiers ordinaires. Toutefois, dans leur cas, les informations ne sont pas conservées par le système de fichiers mais transmises directement aux périphériques.

## 1.7 Retour sur l'interpréteur de commandes

Parmi les autres services de base de l'interpréteur de commandes, on rencontre essentiellement:

- la génération de noms de fichiers,
- la redirection des entrées-sorties,
- la possibilité d'exécuter plusieurs programmes en parallèle en les faisant coopérer selon la technique du travail à la chaîne,

- la gestion de variables,
- la substitution de commandes,
- les fichiers de commandes.

### 1.7.1 Expansion des noms de fichiers

Parfois, vous avez besoin de traiter un ensemble de fichiers, mais vous ne voulez pas taper tous les noms, ou voudriez les abrégés. L'interpréteur de commandes fournit ce qu'on appelle l'expansion. Les mécanismes de base sont :

- Le joker "\*" remplace n'importe quel nombre de caractères, même zéro. Dans `~arthur/gels`, `ls *.IMG` affiche tous les fichiers dont le nom se termine par `.IMG`.
- Le joker "?" remplace un unique caractère. `ls gel?.IMG` afficherait `gel1.IMG gel2.IMG`, ...
- Si vous ne voulez remplacer qu'une plage de caractères, donner la fourchette entre crochets. `[ae]` remplace les lettres a et e, `[a-z]` remplace toutes les minuscules.

### 1.7.2 Entrée et sortie standard

Un programme s'exécutant sous UNIX dispose, pour réaliser ses entrées-sorties, d'un certain nombre de *périphériques logiques* repérés par un numéro de 0 à N.

Trois sont particularisés:

- Le périphérique logique 0 a pour vocation de réaliser des entrées, on le nomme *entrée standard*, il est affecté par défaut au clavier du terminal.
- Le périphérique logique 1 a pour vocation de réaliser des sorties, on le nomme *sortie standard*, il est affecté par défaut à l'écran du terminal.
- Le périphérique logique 2 a pour vocation d'être le support des messages d'erreurs, on le nomme *erreur standard*, il est affecté par défaut à l'écran du terminal.

### 1.7.3 Redirection

Rediriger signifie assigner un fichier différent aux entrées-sorties standard.

On peut lire l'entrée standard à partir d'un fichier au lieu de lire à partir du clavier, on appelle cela *rediriger* l'entrée standard. Exemple :

```
wc <fichier_entree
```

Le fait d'exécuter une commande suivie du signe < suivi d'un nom de fichier, a pour effet de rediriger l'entrée standard de cette commande à partir du fichier indiqué.

On peut écrire la sortie standard dans un fichier au lieu d'écrire à l'écran, on appelle cela *rediriger* la sortie standard. Exemple :

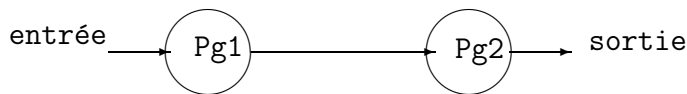
```
ls >fichier_sortie
```

Le fait d'exécuter une commande suivie du signe > suivi d'un nom de fichier, a pour effet de rediriger la sortie standard de cette commande dans le fichier indiqué.

On peut combiner les deux mécanismes et rediriger à la fois l'entrée et la sortie d'un programme.

#### 1.7.4 Les tubes et les filtres

Il arrive très souvent que l'on désire faire coopérer deux programmes de manière à ce que le second effectue son travail sur l'information produite par le premier selon le schéma suivant:



Pour réaliser cela, le système établit un tampon entre les deux programmes, et connecte la sortie du premier programme à l'entrée du tampon, et l'entrée du second programme à la sortie du tampon. Dans le jargon UNIX, un tel tampon porte le nom de *pipe*.

On peut faire coopérer `ls` et `wc` par un fichier réel :

```
ls >fichier
wc <fichier
rm fichier
```

ou utiliser un *pipe*

```
ls | wc
```

Un grand nombre de programmes se contentent de lire un flot de données, de faire un traitement sur ces données et d'écrire un flot de données résultat.

Ces programmes prennent donc leurs données sur l'entrée standard et écrivent leurs résultats sur la sortie standard. Un programme qui respecte cette convention porte (dans le jargon UNIX) le terme de *filtre*.

La commande `grep` est un exemple de filtre, elle ne transmet en sortie que les lignes provenant de son entrée qui contiennent la chaîne donnée en argument.

Exemple :

```
ls | grep projet
```

affichera la liste des fichiers dont le nom contient `projet`.

### 1.7.5 Variables d'environnement

Au cours de la procédure de connexion, l'interpréteur de commandes de lancement définit plusieurs variables d'environnement. Pour afficher leur contenu, utiliser `echo $VARIABLE`. `echo` copie son argument sur la sortie standard, et le signe `$` substitue à la variable sa valeur. En voici quelques unes, parmi les plus importantes :

`TERM` le type de terminal utilisé. Par exemple, `xterm`. Est utilisée par de nombreux programmes pour déterminer les caractéristiques et les caractères de contrôle pour manipuler le terminal.

`DISPLAY` l'écran d'affichage. Important si vous utilisez X (voir le chapitre correspondant).

`PRINTER` l'imprimante utilisée par défaut par la commande `lpr`.

`PATH` la liste des répertoire où l'interpréteur de commandes va aller chercher les commandes.

`HOME` votre répertoire-utilisateur, où vous êtes lors de votre connexion.

Pour changer les variables shell, il faut utiliser `setenv VARIABLE valeur` sous `csh` et `tcsh`.

Pour ajouter un répertoire dans votre `PATH`, par exemple, si arthur veut que son répertoire `bin` y soit, utiliser : `setenv PATH ${PATH}:%HOME/bin` sous `csh` et `tcsh`.

Le répertoire courant est représenté par un point dans le `PATH`.

Vous pouvez mettre ce type de modifications de votre `PATH` dans votre fichier de démarrage. Cela vous évite de taper un chemin d'accès quand vous tapez une commande qui se trouve dans un répertoire défini dans `PATH`. Pour appeler la commande, vous n'avez à taper que le nom du fichier, l'interpréteur de commandes va le chercher dans tous les répertoires jusqu'à ce qu'il le trouve. S'il ne le trouve pas, l'interpréteur de commandes rend le message `command not found`. Pour appeler une commande qui se trouve dans un répertoire qui n'est pas dans le `PATH`, il faut donner le chemin complet (absolu ou relatif). Pour voir tous les répertoires du `PATH`, taper `echo $PATH`.

### 1.7.6 Les alias

Vous voudriez parfois abrégé les commandes interpréteur de commandes que vous utilisez fréquemment. Vous pouvez, pour ce faire, vous définir des alias. Il est bien sûr plus pratique de mettre ces alias dans un fichier de démarrage. Dans l'exemple suivant, nous définissons l'alias `ll` pour la commande `ls -l` :

– `alias ll 'ls -l'` sous `csh`.

– `alias ll='ls -l'` sous `ksh` et `bash`.

L'utilisation des différentes marques de citation dans un interpréteur de commandes n'est pas une chose évidente. Si vous avez des problèmes, regardez les pages de manuel. La forme ci-dessus devrait néanmoins marcher dans la plupart des cas.

### 1.7.7 La substitution de commande

La substitution de commande consiste à écrire une commande entourée du signe ‘ (back quote). On prendra garde à ne pas confondre ce caractère avec le caractère ’ (simple quote). Sur rencontre d’une commande entourée du signe ‘, l’interpréteur de commandes exécute la commande et remplace le texte de la commande par sa sortie (le texte produit par son exécution). Exemple:

```
$ pwd
/users/lgi/systemeV/moi/enseignement
$ D='pwd'
$ echo $D
/users/lgi/systemeV/moi/enseignement
$ ls texinfo*
texinfo texinfo-2 texinfo-4 texinfo-1 texinfo-3 texinfo.texinfo
$ FICS='ls texinfo*'
$ echo $FICS
texinfo texinfo-1 texinfo-2 texinfo-3 texinfo-4 texinfo.texinfo
```

### 1.7.8 Les fichiers de commandes

Il arrive souvent que l’on fasse du travail répétitif nécessitant de taper souvent les mêmes commandes enchaînées. Il est alors agréable de pouvoir mettre une fois pour toutes ces commandes dans un fichier, et d’invoquer ensuite le fichier provoquant ainsi l’exécution des commandes comme si on les avaient tapées au terminal. Un tel fichier est dit “fichier de commandes” et s’appelle un *shell script* dans le jargon UNIX.

### 1.7.9 Commandes diverses

La commande `cat` permet de concaténer et d’afficher un ou plusieurs fichiers.

La commande `more` permet de faire défiler un fichier page par page.

La commande `find` permet de chercher un fichier dans une liste de répertoires.

La commande `grep` permet de chercher une expression dans une liste de fichiers ou de répertoires.

La commande `ps` permet de visualiser les processus de la machine.

La commande `kill` permet de tuer une liste de processus de la machine, vous appartenant.



## Chapitre 2

# L'éditeur de texte vi

vi (pour Visual Editor) est un éditeur de texte pleine page standard disponible sur tous les systèmes UNIX.

Pour lancer l'éditeur, on appelle la commande UNIX vi sur la ligne de commande de l'interpréteur de commandes.

```
% vi nomDeFichier
```

Si le fichier désigné n'existe pas, il est créé et le résultat de l'édition y sera enregistré à la fin de la session. Si le fichier existe, son contenu est affiché ; il peut alors être modifié.

Si tout se passe bien, l'écran suivant apparaît (le fichier *nomDeFichier* n'existait pas avant l'édition) :

```
~
~
"nomDeFichier" [New file] 1 line, 1 char.
```

Le signe ~ en début de ligne indique qu'aucun caractère n'est présent sur la ligne.

L'éditeur fonctionne dans trois modes : le mode commande, le mode insertion et le mode **ex**.

- Le mode *commande* permet de déplacer le curseur dans le fichier, de modifier le texte, ou d'entrer en mode insertion. Les caractères entrés en mode commande restent invisibles à l'écran. C'est le mode initial.
- Le mode *insertion* permet de taper un nouveau texte. Tous les caractères tapés au clavier apparaissent dans le texte.
- Le mode *ex* permet d'entrer des commandes de l'éditeur **ex**.

## 2.1 Passage en mode insertion

À partir du mode *commande*, la touche *i* permet d'entrer en mode *insertion*. Après avoir inséré le texte désiré, le retour en mode *commande* s'effectue par la touche *ESC*

Cette touche se trouve tout en haut à gauche du clavier.

Les déplacements du curseur en mode insertion sont interdits; seule la touche *BackSpace* efface le dernier caractère inséré précédemment.

Les touches *a*, *A*, *I*, *o* et *O* permettent aussi de passer en mode *insertion*.

## 2.2 Le mode commande

**Déplacement du curseur en mode commande** Un certain nombre de commandes sont utilisées pour déplacer le curseur dans un texte.

On utilise les touches *i*, *j*, *k*, *l*, *w*, 0 (zéro), \$, *b*, +, −.

**Suppression d'un texte** Des commandes permettent de supprimer des portions de textes.

Ce sont les commandes *x*, *dd*, *dw*, *D*.

**Remplacement d'un texte** Les commandes de remplacement sont diverses :

*s* (substitution),

*r*, *R* (remplacement),

*cc*, *c0*, *c\$*, *cw*, *C* (changement).

**Répétition des commandes** En règle générale, la commande précédente peut être répétée en appuyant sur la touche *.* (point) *.* Une autre technique consiste à faire précéder une commande par un nombre.

**Annulation d'une commande** Les conséquences de la dernière commande effectuée sont annulées par la touche *u*.

**Sortie et enregistrement** Pour terminer une session d'édition, il suffit de taper la commande *ZZ*. Le fichier est alors enregistré et la main est rendue à l'interpréteur de commande UNIX.

## 2.3 Le mode ex

On distingue un mode *ex* où certaines commandes héritées de l'éditeur *ex* doivent être précédées par le caractère *' :'*.

# Chapitre 3

## L'éditeur de texte *ex*

*ex* est un éditeur de texte standard disponible sur tous les systèmes UNIX en mode ligne.

### 3.1 Les commandes *ex*

Une commande *ex* consiste en une adresse de lignes, suivie d'une commande à appliquer sur les lignes sélectionnées par l'adresse.

#### 3.1.1 Symboles d'adressage des lignes

.	ligne courante
\$	dernière ligne du fichier
%	toutes les lignes du fichier (équivalent à 1,\$)
+ dep	
- dep	déplacement relatif par rapport à une ligne
/motif/	première occurrence du motif (décrit par une expression régulière). La recherche est effectuée par numéro de ligne croissant
?motif?	première occurrence du motif. La recherche est effectuée par numéro de ligne décroissant

<b>Exemples</b>	11	la ligne 11
	.,\$	de la ligne courante à la fin du fichier
	10,.	de la ligne 10 à la ligne courante
	.,+10	10 lignes à partir de la ligne courante
	.,/a[bc]a/	de la ligne courante à la première ligne contenant le mot <i>aba</i> ou le mot <i>aca</i>

### 3.1.2 Les commandes

r fichier	inclusion d'un fichier
w fichier	enregistrement d'un fichier
q	sortie de l'éditeur
wq	enregistrement et sortie de l'éditeur
e fichier	lecture d'un nouveau fichier
d	supprimer une ligne
m adresse	déplacer les lignes sélectionnées vers l'adresse
j	concatène les lignes sélectionnées
s/ancien/nouveau	remplace le premier motif <i>ancien</i> de la ligne par le motif <i>nouveau</i>
s/ancien/nouveau/g	remplace tous les motifs <i>ancien</i> de la ligne par le motif <i>nouveau</i>
s/ancien/nouveau/c	
s/ancien/nouveau/gc	idem, mais avec une demande de confirmation des changements
g/motif/commande	sélectionne un ensemble de ligne contenant le motif, puis applique la commande sur ces lignes.

## 3.2 Les expressions régulières

Les expressions régulières décrivent un langage sur un alphabet  $V$ , c'est à dire un ensemble d'*énoncés bien formés* d'un point de vue syntaxique. Les expressions régulières utilisent trois opérations :

- l'union [...]

*pour*  $V = \{a, b\}$ ,  $L_1 = [ab]$  reconnaît les mots  $a$  et  $b$

- la concaténation

*pour*  $V = \{a, b\}$ ,  $L_2 = ab$  reconnaît le mot  $ab$

- l'itération \*

*pour*  $V = \{a\}$ ,  $L_3 = a^*$  reconnaît, entre autres, les motifs (*vide*),  $a$ ,  $aa$ , ou  $aaaaaaaa$

L'expression régulière  $a^*b^*$  décrit par exemple un langage reconnaissant le motif (*vide*), les motifs composés uniquement de  $a$  ou de  $b$ , et les motifs commençant par un nombre quelconque de  $a$  suivi d'un nombre quelconque de  $b$ .  $a[cd]a$  reconnaît les motifs  $aca$  et  $ada$ .

En supplément des opérations fondamentales, *ex* propose des opérateurs complémentaires utilisables dans les expressions régulières (voir l'annexe).

# Bibliographie

[Rif94] Rifflet (J.M). – *La programmation sous UNIX*. – EDISCIENCE international, 1994.