

I.U.T. Amiens
Département Informatique
Année Universitaire 2002/2003



Le système Unix

C. Drocourt – Màj le 30/09/2003



Le système Unix

30/09/2003

Sommaire

1 - Introduction au système Unix	5
1.1 - Les versions majeures	5
1.2 - Qu'est-ce qu'Unix ?	5
1.2.1 - Système d'exploitation	5
1.2.2 - Unix est multi-tâches	5
1.2.3 - Unix est multi-utilisateurs	5
1.3 - Principales fonctionnalités d'Unix	5
1.3.1 - Gestion des ressources de l'ordinateur	5
1.3.2 - Gestion des données	5
1.3.3 - Communication entre utilisateurs	6
1.3.4 - Environnement de développement	6
1.4 - Le noyau et le reste	6
1.5 - Les interfaces utilisateur	6
1.5.1 - Pour terminaux alpha-numériques	6
1.5.2 - Pour terminaux graphiques	6
1.6 - Les acteurs du monde Unix	6
1.6.1 - L'utilisateur	6
1.6.2 - L'administrateur, ou super-utilisateur, ou root	7
1.6.3 - Le programmeur	7
1.7 - Comptes et accès	7
1.7.1 - Introduction	7
1.7.2 - Fichier passwd	7
1.7.3 - L'administrateur du système	8
1.7.4 - Le répertoire privé « home directory »	8
1.7.5 - Ouvrir une session « logging in »	9
1.7.6 - Fermer une session de travail « logging out »	9
1.7.7 - Les différents types de connexion	9
2 - Le système de fichiers	11
2.1 - Présentation générale	11
2.2 - Hiérarchie classique d'un serveur Unix	11
2.3 - I-nombre et i-nœud	11
2.4 - Les types de fichier	12
2.4.1 - Fichiers ordinaires	12
2.4.2 - Fichiers catalogue	12
2.4.3 - Fichiers spéciaux	12
2.4.4 - Fichiers lien symbolique, fichiers tube nommé, fichiers socket	12
2.5 - Les principales commandes de manipulation de fichiers	12
2.5.1 - Référence du catalogue de travail : pwd	12
2.5.2 - Changement du catalogue de travail : cd	12
2.5.3 - Contenu d'un catalogue et caractéristiques d'un fichier : ls	13
2.5.4 - Copie physique d'un fichier : cp	14
2.5.5 - Déplacement ou changement du nom d'un fichier : mv	14
2.5.6 - Suppression d'un fichier : rm	14



Le système Unix

30/09/2003

2.5.7 - Création d'un catalogue : mkdir	14
2.5.8 - Effacement d'un catalogue vide : rmdir	15
2.5.9 - Effacement d'un catalogue non vide : rm -r	15
2.5.10 - Création de lien sur un fichier : ln	15
2.6 - Mode d'un fichier	15
2.6.1 - Le principe	15
2.6.2 - Modification des droits d'accès: la commande chmod	16
2.6.3 - Changement de propriétaire et de groupe propriétaire	17
2.7 - Les bits spéciaux	17
2.7.1 - Le sticky-bit	17
2.7.2 - Le set-uid bit	17
3 - Les principales commandes du système	19
3.1 - Deux type de commandes	19
3.1.1 - Les commandes externes	19
3.1.2 - Les commandes internes	19
3.2 - Code de retour d'une commande (exit status)	19
3.3 - Fichiers standards d'E/S	19
3.3.1 - Définition	19
3.3.2 - Les redirections	19
3.4 - Les filtres	21
3.5 - Les tubes	21
3.6 - D'autres commandes Unix	22
3.6.1 - Identification par le système	22
3.6.2 - Manipulation de fichiers et espace disque	22
3.6.3 - Commandes d'impression	23
4 - La gestion des processus	24
4.1 - Notion de processus	24
4.2 - Types de processus	24
4.2.1 - Processus "utilisateur"	24
4.2.2 - Processus "système"	24
4.3 - L'ordonnanceur de tâches	24
4.4 - Caractéristiques d'un processus	25
4.4.1 - Bloc de contrôle	25
4.4.2 - Calcul de la priorité d'un processus	25
4.4.3 - La commande nice	25
4.5 - Affichage des caractéristiques d'un processus	25
4.6 - Processus en avant plan	26
4.7 - Processus en arrière plan	26
4.8 - Le contrôle de jobs	26
4.9 - Les signaux asynchrones	27
4.9.1 - Les signaux	27
4.9.2 - La commande kill	27
5 - L'éditeur de texte VI	28
5.1 - Etude rapide	28
5.1.1 - variable d'environnement TERM	28
5.1.2 - Entrée et sortie de vi	28
5.1.3 - Le mode commande	28



Le système Unix

30/09/2003

5.1.4 - Le mode insertion	29
5.2 - Configuration de vi	29
5.2.1 - En mode commande	29
5.2.2 - A la connexion	30
5.2.3 - A chaque invocation	30
5.3 - Avantages et inconvénients	30
5.4 - Compléments	31
5.4.1 - Appel d'une commande UNIX	31
5.4.2 - Utiliser des buffers nommés	31
5.4.3 - Filtrer du texte	31
6 - L'interpréteur de commande bash	32
6.1 - Qu'est-ce qu'un shell ?	32
6.2 - Configuration du shell bash	32
6.3 - Fichier de commandes (script-shell)	32
6.4 - Les variables du shell	33
6.4.1 - Variables locales	33
6.4.2 - Variables d'environnement (variables exportées)	33
6.4.3 - Typer une variable	33
6.4.4 - Quotage des variables	34
6.4.5 - Variables de substitution	34
6.5 - Les alias	34
6.6 - Masque des droits d'accès	34
6.7 - Traitement des interruptions	34
6.8 - Les instructions du shell	34
6.8.1 - Commandes de test	34
6.8.2 - Structure de contrôle	35



1 - Introduction au système Unix

1.1 - Les versions majeures

- ❑ System V d'AT&T
- ❑ BSD (Berkeley Software Distribution)

Autres versions répandues

- ❑ AIX d'IBM
- ❑ Sun OS, Solaris
- ❑ Linux

1.2 - Qu'est-ce qu'Unix ?

Système d'exploitation **multi-tâches** et **multi-utilisateurs**.

1.2.1 - Système d'exploitation

Assure aux différentes tâches et aux différents utilisateurs une bonne répartition des ressources de l'ordinateur : mémoire, processeur, disque, imprimante, terminaux, réseau...

1.2.2 - Unix est multi-tâches

Plusieurs processus s'exécutent « en même temps » sur le même ordinateur : des processus système (ordonnanceur de tâches, gestionnaire d'impression...) et des processus utilisateur qui exécutent le code des commandes lancées par des utilisateurs du serveur Unix.

1.2.3 - Unix est multi-utilisateurs

Pour se connecter au serveur Unix, un utilisateur doit disposer d'un nom de login et d'un mot de passe qui lui sont alloués par l'administrateur du serveur. Plusieurs utilisateurs peuvent lancer simultanément plusieurs processus.

1.3 - Principales fonctionnalités d'Unix

1.3.1 - Gestion des ressources de l'ordinateur

Sous Unix, le temps d'utilisation du processeur de l'ordinateur est réparti entre différentes tâches ce qui se traduit par l'exécution simultanée de plusieurs programmes lancés par plusieurs utilisateurs ; tout ceci étant totalement transparent pour les utilisateurs.

1.3.2 - Gestion des données

Celle-ci consiste en l'organisation, la maintenance et l'accès aux unités de stockage (mémoire, disque, bandes...). Unix étant multi-utilisateurs, il doit assurer la sécurité et la confidentialité des données : ceci est réalisé par la gestion du système de fichiers.



Le système Unix

30/09/2003

1.3.3 - Communication entre utilisateurs

Le système Unix est conçu pour communiquer. Courrier électronique, News, transfert de fichiers, connexion distante...

1.3.4 - Environnement de développement

Conçu à l'origine par des développeurs pour des développeurs, Unix possède en standard éditeurs de texte, compilateurs, analyseurs de syntaxe... et autres outils d'aide à la programmation.

1.4 - Le noyau et le reste...

Le **noyau** est chargé du rôle essentiel d'un système d'exploitation : assurer la gestion de la mémoire, la gestion des entrées-sorties de bas niveau et l'enchaînement des différentes tâches.

Un processus accède aux services du noyau par le biais **d'appels système**.

La norme **POSIX** définit un ensemble de services que doit offrir un système d'exploitation et la façon dont on accède à ces services.

1.5 - Les interfaces utilisateur

1.5.1 - Pour terminaux alpha-numériques

Unix reconnaît un nombre impressionnant de terminaux de ce type, dont les définitions sont regroupées sous `/usr/lib/terminfo`. La variable d'environnement `TERM` contient une chaîne de caractères qui définit le terminal utilisé : « `TERM=vt220` » renvoie ainsi à « `/usr/lib/terminfo/v/vt220` ».

1.5.2 - Pour terminaux graphiques

X-Window est un protocole de gestion de ces terminaux, connu sous le nom de X11, sur lequel se basent des interfaces plus sophistiquées telles que Motif.

1.6 - Les acteurs du monde Unix

1.6.1 - L'utilisateur

Après avoir décliné son identité composée d'un nom de connexion (**LOGNAME**) et d'un mot de passe, l'utilisateur dispose d'un shell (interpréteur de commande) et se trouve positionné dans un répertoire qui en général est privé (**HOME**). Au login, un certain nombre de fichiers administratifs sont utilisés pour établir l'identité de l'utilisateur : `/etc/passwd` (définition des utilisateurs) et `/etc/group` (définition des groupes d'utilisateurs ; chaque utilisateur fait partie d'un groupe de travail) sont les plus connus. L'utilisateur travaille avec des shells et des utilitaires ; il peut aussi regrouper des commandes dans des fichiers appelés script-shells dont il pourra lancer plus tard l'exécution. Chaque fichier Unix étant protégé par des droits d'accès, l'utilisateur est limité dans ses actions : il ne peut par exemple décider de l'arrêt du serveur.



Le système Unix

30/09/2003

1.6.2 - L'administrateur, ou super-utilisateur, ou root

C'est un utilisateur qui a tous les droits... ou presque. C'est à lui de veiller au bon fonctionnement du serveur Unix, de créer des comptes pour les nouveaux utilisateurs, d'effectuer des sauvegardes, d'installer des nouveaux périphériques...

1.6.3 - Le programmeur

C'est un utilisateur qui crée de nouvelles commandes Unix à partir de sources écrites dans un langage évolué comme le langage C.

1.7 - Comptes et accès

1.7.1 - Introduction

Pour utiliser un ordinateur sous Unix, une personne a besoin d'un compte. Un compte permet à l'utilisateur d'accéder aux ressources disponibles. De plus, le compte identifie l'utilisateur sur le système. Il est nécessaire que les utilisateurs possèdent un compte dans un contexte multi-utilisateurs afin de gérer l'accès aux ressources. Au minimum, un compte est constitué d'un répertoire privé et d'une entrée dans le fichier `/etc/passwd`.

Ce chapitre présente les notions de base associées aux accès par l'intermédiaire d'un compte. Il donne, entre autres, des informations sur l'administrateur du système. C'est le responsable des ordinateurs. Ensuite, on peut voir le concept de répertoire privé associé à un compte.

1.7.2 - Fichier passwd

La majorité des informations concernant un utilisateur se retrouve dans un fichier qui se nomme « `passwd` ». Ce fichier se trouve dans le répertoire `/etc`. Comme exemple, prenons un utilisateur qui s'appellerait « `laforge` ». Voici la ligne dans le fichier `/etc/passwd` qui correspondrait à son compte:

```
laforge:VByALL2dcWRFA:501:100:Laforge Normand:/home/laforge:/bin/tcsh
```

ou

```
laforge:x:501:100:Laforge Normand:/home/laforge:/bin/bash
```

Cette entrée est divisée en champs séparés par des caractères « `:` » dont voici la signification.

`laforge` : c'est le nom d'utilisateur (« `login name` »). Ce champ est utilisé par le système pour reconnaître la personne et ainsi contrôler ses permissions d'accès à des fichiers ou des services. Contrairement à la vie réelle, ce nom doit être unique sur la machine ou même sur le réseau local dans la plupart des cas. Le détenteur d'un compte doit être identifié sans ambiguïté par le système pour des raisons de sécurité, de gestion et de communication. Par convention, le nom d'utilisateur est toujours formé de lettres minuscules et/ou de chiffres.

`VByALL2dcWRFA` : il s'agit d'une représentation encodée du mot de passe (« `password` ») du compte en question. Celui que l'utilisateur entre au clavier est encodé à l'aide d'une fonction (dont l'inverse est supposément impossible à établir...) et comparé avec ce champ. Donc pour accéder au compte de « `laforge` » sur la machine en question, il suffit de connaître son nom d'utilisateur et son mot de passe. Il est à noter que le système fait la distinction entre les majuscules et les minuscules donc



Le système Unix

30/09/2003

les lettres « A » et « a » ne sont pas les mêmes lettres lorsqu'on tape son mot de passe au clavier. Sur les version plus récentes d'unix, pour des raisons de sécurité, le mot de passe encodé se trouve dans un autre fichier et il y a un x a la place de ce champ, c'est ce qu'on appelle les « shadow password ».

501 : c'est le numéro d'identification de l'utilisateur (« user id ») qui est unique. Il est utile au système, car ce dernier fait correspondre ce numéro à un compte pour gérer les accès. Par exemple, dans les inodes du système de fichiers, on retrouve ce numéro dans le champ qui représente le propriétaire.

100 : chaque usager fait partie d'un groupe. Ce concept est utile quand on veut donner certaines permissions particulières à des personnes qui travaillent sur un projet et qui doivent avoir accès à certains fichiers, par exemple. La notion de groupe permet de classer les types d'utilisateurs avec les permissions qui s'y rattachent. Le système Unix fait correspondre le compte à un groupe avec ce numéro. Ce chiffre est celui qui est présent dans les inodes et qui représente le groupe auquel le fichier appartient.

Laforge Normand : c'est le nom du détenteur du compte. Ce champ peut également contenir d'autres informations comme le bureau de l'utilisateur (son numéro, par exemple), son numéro de téléphone au bureau et à la maison. Ces sous-champs seraient alors séparés par des virgules.

/home/laforge : c'est le répertoire privé de l'utilisateur. C'est aussi le répertoire de travail par défaut lorsqu'on ouvre une session de travail « logging in ».

/bin/tcsh ou /bin/bash : pour terminer, le dernier champ représente l'interpréteur de commandes par défaut (« login shell ») qui est lancé à l'ouverture d'une session de travail.

L'organisation du fichier /etc/passwd peut être différente sur certaines machines où le système NIS est installé. Ces systèmes ne seront pas expliqués mais servent respectivement à augmenter la sécurité et à distribuer l'information sur un réseau.

1.7.3 - L'administrateur du système

Toutes les machines Unix ont une entrée dans le fichier /etc/passwd qui correspond à l'utilisateur « root ». C'est l'administrateur du système. Il ne faut pas le confondre avec le répertoire qui est parent de tous les fichiers et qui porte aussi le nom de « root », représenté par le caractère « / ». L'utilisateur « root » a tous les droits possibles sur l'ordinateur et peut créer ou détruire des fichiers ou des comptes entiers. Il est aussi la personne ressource en cas de problèmes comme l'oubli d'un mot de passe, par exemple. C'est aussi lui qui fait la sauvegarde des données, l'installation de logiciels, etc.

1.7.4 - Le répertoire privé « home directory »

Comme mentionné plus haut, tous les comptes sont associés à un répertoire privé. L'utilisateur est libre de gérer son répertoire comme il l'entend. Il peut ainsi créer une arborescence de fichiers et de répertoires à partir de son répertoire personnel. L'espace disque utilisé peut être limité par la capacité du disque ou par un système de quota. Un système de quota est un outil de gestion qui existe pour empêcher les abus et qui est géré par l'utilisateur « root ». L'administrateur peut ainsi (grâce au quota) décider que l'espace disque maximal utilisé par un utilisateur précis sera d'un certain nombre d'octets.



Le système Unix

30/09/2003

1.7.5 - Ouvrir une session « logging in »

Pour ouvrir une session, l'utilisateur doit entrer son nom d'utilisateur et mot de passe. Voici un exemple d'ouverture de session par l'utilisateur « heidi ».

```
Red Hat Linux release 6.1 (Cartman)
Kernel 2.2.14 on an i686
login: heidi
Password:
Last login: Mon Nov 6 15:45:43 from rahan.iut.fr
*****
* Bienvenue pour cette première session sous Unix :      *
*   Serveur etud                                       *
*   interpréteur de commande : bash                     *
*****
You have new mail.
[heidi@etud heidi]$
```

Noter que le mot de passe n'est pas affiché à l'écran. Heidi a entré son nom « heidi » et a appuyé sur la touche ENTER. Quand le système a affiché le mot « Password », elle l'a entré et a appuyé à nouveau sur la touche ENTER. À ce moment, le système lui donne de l'information sur sa dernière session et affiche le message du jour (ce message est contenu dans le fichier /etc/motd). Pour terminer, heidi reçoit l'invite de commande (« prompt ») lui indiquant que le système est prêt et en attente.

Si l'utilisateur fait une erreur lorsqu'il entre son mot de passe ou son nom d'utilisateur, l'ordinateur affichera un message du genre « Login Incorrect ». En fait, ce message indique que le système est incapable de trouver de ligne du fichier /etc/passwd correspondant à ces deux informations (nom d'utilisateur et mot de passe). À ce moment, il faut réessayer quelques fois en s'assurant de respecter les majuscules et les minuscules. Si le problème persiste, il faut consulter la personne en charge (« root »).

1.7.6 - Fermer une session de travail « logging out »

Avant de quitter un ordinateur, il faut TOUJOURS fermer sa session de travail. Pour cela, il suffit d'entrer la commande « logout ».

Il est certain que votre session de travail est fermée lorsque vous voyez que Unix invite une personne à ouvrir une session avec le mot « login: ». Oublier de fermer sa session pourrait avoir des conséquences graves au point de vue sécurité. Il est aussi possible dans plusieurs cas d'entrer d'autres commandes comme « exit » ou CTRL-d pour terminer une session.

1.7.7 - Les différents types de connexion

Pour se connecter à un serveur unix, diverses possibilités sont envisageables, tout d'abord en mode texte :

- Directement sur la machine en mode console : possible pour une installation personnelle.
- Sur un terminal texte.
- En utilisant une connexion distante en mode texte (telnet ou ssh).

En mode graphique :



Le système Unix

30/09/2003

-
- Directement sur la machine (cas d'une station de travail).
 - A distance, via une station X ou à travers un émulateur X sous windows comme exceed.

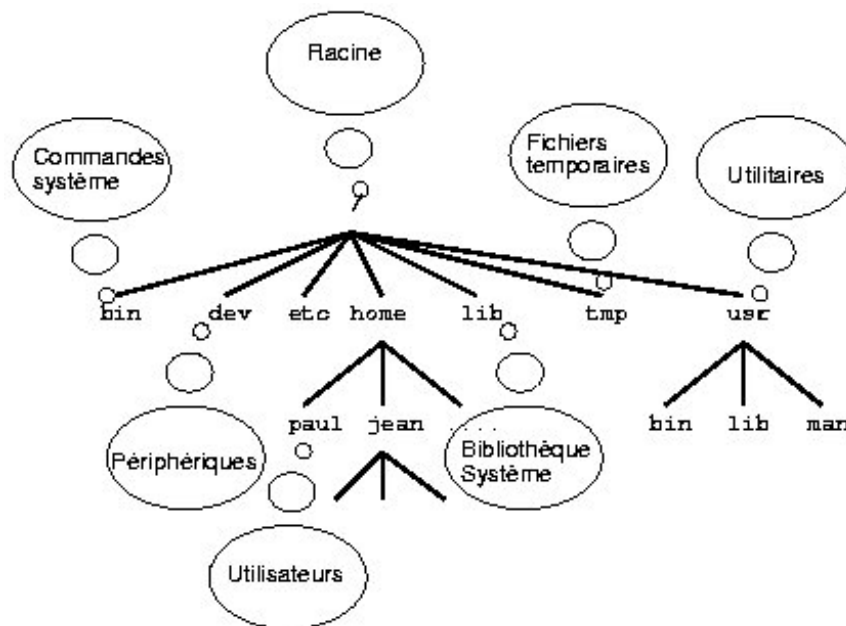
2 - Le système de fichiers

2.1 - Présentation générale

- ❑ Un fichier Unix est une séquence d'octets.
- ❑ A chaque fichier Unix est associé un bloc d'information appelé i-nœud qui est référencé par un i-nombre.
- ❑ La désignation d'un fichier se fait par l'intermédiaire de catalogues (ou répertoire ou directory).

2.2 - Hiérarchie classique d'un serveur Unix

Sous Unix, les fichiers sont accessibles sous la forme d'une arborescence. La haut de cette arborescence est appelée la racine du système de fichier. Ainsi, lorsque que l'on veut faire référence à un fichier, on peut le faire soit en précisant le chemin relatif à l'endroit où l'on se trouve (relatif) soit par rapport à la racine du système (mode absolu). Il n'y a pas comme sous dos ou windows de lecteur car les différents périphériques font parti de cette arborescence.



2.3 - I-nombre et i-nœud

Un i-nœud est référencé par un unique i-nombre qui est associé à un ou plusieurs fichiers. Ce i-nœud contient:

- ❑ la taille du fichier en octet
- ❑ les adresses des blocs utilisés sur le disque dur
- ❑ le numéro d'identification (UID) du propriétaire
- ❑ le numéro d'identification (GID) du groupe propriétaire
- ❑ le nombre de liens : le nombre de fois où le i-nombre référençant le i-nœud est associé à un nom de fichier
- ❑ le type (ordinaire, catalogue, spécial...)



Le système Unix

30/09/2003

- ❑ le droit d'accès au fichier
- ❑ la date de dernier accès
- ❑ la date de dernière modification du i-nœud.

2.4 - Les types de fichier

2.4.1 - Fichiers ordinaires

Sur disque, ils peuvent contenir des données, des sources dans un langage donné, des binaires exécutables...

2.4.2 - Fichiers catalogue

Contient une suite de couples (nom, i-nombre). Ils sont également appelés indifféremment répertoires ou directory.

2.4.3 - Fichiers spéciaux

Ils correspondent aux dispositifs d'E/S physiques et sont tous référencés dans le catalogue */dev*.

Ex: */dev/lp0* est la référence d'une imprimante.

/dev/hd9 est la référence d'un disque logique.

Du point de vue de l'utilisateur, il n'y a aucune différence entre ces fichiers et les fichiers ordinaires.

Ex : *cal > /dev/tty12* affiche le calendrier sur l'écran du terminal *tty12*.

2.4.4 - Fichiers lien symbolique, fichiers tube nommé, fichiers socket

Qui correspondent à des éléments particuliers qui seront vu plus loin dans ce cours.

2.5 - Les principales commandes de manipulation de fichiers

2.5.1 - Référence du catalogue de travail : pwd

Cette commande permet de connaître sa position dans le système de fichier, exemple :

```
[roger@etud roger]$ pwd
/home/etud/roger
[roger@etud roger]$
```

2.5.2 - Changement du catalogue de travail : cd

Pour se déplacer dans l'arborescence du système de fichier, on utilise la commande **cd**. On peut indiquer le chemin dans lequel on veut se déplacer de façon relative (la position courant est donnée par un point et le parent par un *..*) ou absolue (dans lequel on précise que l'on fait référence à la racine avec un */*). De plus, **cd** sans argument retourne au répertoire de connexion. Exemples :



Le système Unix

30/09/2003

```
[roger@etud roger]$ pwd
/home/etud/roger
[roger@etud roger]$ cd ..
[roger@etud roger]$ pwd
/home/etud
[roger@etud roger]$ cd /tmp
[roger@etud roger]$ pwd
/tmp
[roger@etud roger]$ cd ../home
[roger@etud roger]$ pwd
/home
[roger@etud roger]$ cd
[roger@etud roger]$ pwd
/home/etud/roger
[roger@etud roger]$
```

2.5.3 - Contenu d'un catalogue et caractéristiques d'un fichier : ls

Pour afficher la liste des fichiers et répertoires (catalogues ou directory) on utilise la commande **ls**, exemple :

```
[roger@etud roger]$ ls
file1 file2 repl
[roger@etud roger]$
```

ls admet certaines options :

- a** : tous les fichiers, y compris les fichiers cachés (fichiers dont le nom commence par un point ".")
- F** : identifie les fichiers en ajoutant "/" aux noms de dir., "*" aux exécutable, "@" aux liens symboliques
- R** : liste en parcourant récursivement tous les sous-répertoires
- i** : indique le i-nombre
- ld répertoire** : affiche informations sur les répertoires spécifiés (et non pas sur leur contenu)
- l** : listage long, cf ci-dessous

-	rwxrwxrwx	1	root	adm	2345	janv 20	12:56	/tmp/fic1
Type de fichier	Doits utilisateurs	nb liens	Propriétaire	groupe	taille	date de modification	heure	nom du fichier
a	b	c	e	f	e	f	g	h

- **a** type de fichier
 - Fichier ordinaire
 - d Répertoire
 - l Lien symbolique
 - c Fichier spécial en mode caractères
 - s Socket
 - p Tube nommé
- **b** droits d'accès
- **c** Nombre de lien sur ce fichier
- **d** Le nom du propriétaire du fichier
- **e** Le nom du groupe auquel appartient ce fichier
- **f** et **g** La date et éventuellement l'heure de dernière modification (quand il s'agit de l'année actuelle)



Le système Unix

30/09/2003

- **h** Le nom du fichier

2.5.4 - Copie physique d'un fichier : cp

La commande **cp** réalise la copie physique d'un fichier, c'est-à-dire une duplication des données sur le disque dur. Exemple :

```
[roger@etud roger]$ ls
file1 file2  rep1
[roger@etud roger]$ cp file1 file3
[roger@etud roger]$ ls
file1 file2 file3  rep1
[roger@etud roger]$
```

2.5.5 - Déplacement ou changement du nom d'un fichier : mv

Cette commande permet en fait uniquement de déplacer un fichier mais permet de donner un nom différents au fichier destination, elle peut donc être utilisée pour renommer un fichier.

```
[roger@etud roger]$ ls
file1 file2 file3  rep1
[roger@etud roger]$ ls /tmp
[roger@etud roger]$ mv file2 /tmp
[roger@etud roger]$ ls /tmp
file2
[roger@etud roger]$ ls
file1 file3  rep1
[roger@etud roger]$ mv file3 file4
[roger@etud roger]$ ls
file1 file4  rep1
[roger@etud roger]$
```

2.5.6 - Suppression d'un fichier : rm

Cette commande permet de supprimer un fichier dans le répertoire courant ou ailleurs dans l'arborescence en précisant le chemin d'accès. Exemple :

```
[roger@etud roger]$ ls /tmp
file2
[roger@etud roger]$ rm /tmp/file2
[roger@etud roger]$ ls /tmp
[roger@etud roger]$
```

2.5.7 - Création d'un catalogue : mkdir

Cette commande permet la création d'un repertoire. Exemple :

```
[roger@etud roger]$ ls
file1 file4  rep1
[roger@etud roger]$ mkdir rep2
[roger@etud roger]$ ls
file1 file4  rep1  rep2
[roger@etud roger]$
```



Le système Unix

30/09/2003

2.5.8 - Effacement d'un catalogue vide : rmdir

Attention il faut impérativement que le répertoire soit vide sinon le système refusera !! Exemple :

```
[roger@etud roger]$ ls
file1 file4 rep1 rep2
[roger@etud roger]$ rmdir rep2
[roger@etud roger]$ ls
file1 file4 rep1
[roger@etud roger]$ rmdir rep1
rmdir: `rep1': Le répertoire n'est pas vide.
[roger@etud roger]$ ls
file1 file4 rep1
[roger@etud roger]$
```

2.5.9 - Effacement d'un catalogue non vide : rm -r

Exemple :

```
[roger@etud roger]$ ls
file1 file4 rep1
[roger@etud roger]$ rm -r rep1
[roger@etud roger]$ ls
file1 file4
[roger@etud roger]$
```

2.5.10 - Création de lien sur un fichier : ln

La commande *ln* réalise un lien normal, c'est-à-dire une nouvelle référence à un fichier physique. Il n'y a pas création d'un i-nœud ni duplication des données sur le disque dur. Pour réaliser un lien normal entre deux fichiers, ceux-ci doivent appartenir au même système de fichier.

La commande *ln -s* permet de créer un lien symbolique : il y a création d'un fichier de type lien symbolique dont le contenu est la référence du fichier à lier, c'est ce type de liens qui est le plus couramment utilisé.

Exemple :

```
[roger@etud roger]$ ls -l
total 0
-rw-r--r--  1 roger  adm    0 déc  2 16:04 file1
-rw-r--r--  1 roger  adm    0 déc  2 16:19 file4
[roger@etud roger]$ ln -s file1 file2
[roger@etud roger]$
[roger@etud roger]$ ls -l
-rw-r--r--  1 roger  adm    0 déc  2 16:04 file1
lrwxrwxrwx  1 roger  adm    5 déc  2 16:41 file2 -> file1
-rw-r--r--  1 roger  adm    0 déc  2 16:19 file4
[roger@etud roger]$
```

2.6 - Mode d'un fichier

2.6.1 - Le principe

Chaque utilisateur possède un numéro d'identification (UID) et appartient à un groupe donné.

```
$ id
```



Le système Unix

30/09/2003

uid=172(gaston) gid=102(gaffeur)

Pour un fichier donné, les utilisateurs sont classés en trois catégories:

- Le propriétaire
- Le groupe propriétaire
- Les autres

Chacune de ces catégories peut avoir trois droits:

- le droit r (lecture)
- le droit w (écriture)
- le droit x (execution)

Selon le type de fichier, ces droits ont une signification particulière.

Droits	Fichier	Répertoire
r	voir le contenu	voir le contenu
w	modifier ou effacer le contenu	créer ou détruire des fichiers
x	exécuter	traverser

Exemples :

rWX	rWX	rWX	rW-	r - -	r - -
user	group	other	user	group	other
<i>L'utilisateur, le groupe, les autres ont tous les droits</i>			<i>L'utilisateur ne peut pas exécuter Le groupe et les autres ne peuvent que lire</i>		

2.6.2 - Modification des droits d'accès: la commande chmod

Change les droits d'accès à un fichier ou à un répertoire.

mode symbolique

chmod [-R-f] [ugo|a]{{-|[rwxst]]}|{=[rwxst]}}{Fich...|Rep...}

mode numérique ou absolu

chmod [-R-f] Code_accès {Fich...|Rép...}

La notation **symbolique** peut se résumer de la manière suivante :

- qui=permission*
- qui+permission* :ajouter droit(s)
- qui-permission* :enlever droit(s)

avec

- qui* = {u}{g}{o}{a}
- permission* = {r}{w}{x}

La notation **octale** de type : *ugo* où *u*, *g*, *o* sont des valeurs octales de 0 à 7 définissant les droits d'accès à l'égard de l'utilisateur, du groupe et des autres (**others**) selon la règle :

- 0 : aucun droit
- 1 : exécution (x)
- 2 : écriture (w)
- 4 : lecture (r)
- addition* = combinaison

Exemple : Pour instaurer le droit *rxr-x---* au fichier *beurk*, on tape `chmod 750 beurk` car $7=1(x)+2(w)+4(r)$, $5=1(x)+4$, $0=0(\text{aucun droit})$



Le système Unix

30/09/2003

2.6.3 - Changement de propriétaire et de groupe propriétaire

`chown <proprio> <fic>`

`chgrp <proprio> <fic>`

Ces changements ne peuvent être effectués que par le propriétaire initial du fichier ou le root.

2.7 - Les bits spéciaux

2.7.1 - Le sticky-bit

Ce bit désigné par `t` apparaît à la place du champ `x` dans le groupe de bits concernant la classe `other`. Il ne s'applique qu'aux fichiers exécutables et aux répertoires.

Signification pour un fichier exécutable

Cet attribut indique que le segment texte d'un fichier exécutable n'est swappé out qu'une seule fois et qu'il est conservé dans l'espace disque de swap une fois que la commande est terminée. Ce mécanisme réduit donc le nombre de swap out et permet un chargement en mémoire plus rapide lors d'une exécution ultérieure d'une commande.

Signification pour un répertoire

Pour un répertoire, cette possibilité a été introduite par la version BSD 4.3. Notons qu'elle est actuellement implantée dans la totalité des versions commerciales. Pour un répertoire public dont les permissions sont `drwxrwxrwt`, le bit `t` a pour effet de protéger les fichiers qu'il contient contre la suppression. Cela signifie que dans ce cas, seul le propriétaire du fichier et root peuvent supprimer un fichier situé dans un tel répertoire.

Cette notion a été introduite pour protéger contre la suppression les fichiers temporaires qui se trouvent dans les répertoires publics comme `/usr/tmp`, ou `.usr/spool/tmp`, ...

2.7.2 - Le set-uid bit

On le positionne pour un binaire exécutable pour "prêter" les droits d'accès du propriétaire du fichier aux autres utilisateurs pendant le temps d'exécution du processus exécutant le code contenu dans le fichier.

Cette fonctionnalité est notamment utilisée pour permettre à un utilisateur de changer son mot de passe.

```
$ ls -l /etc/passwd /usr/bin/passwd
-rw-r--r-- 1 root system 10437 Jun 11 16:14 /etc/passwd
-r-sr-x--x 1 root security 48750 Oct 26 1994 /usr/bin/passwd
```

Le fichier `/etc/passwd` contient les définitions des utilisateurs, il est propriété de l'utilisateur `root` et du groupe d'utilisateur `system` et il n'est pas accessible en écriture à l'utilisateur `Gaston` du groupe `gaffeur`.

Le fichier `/usr/bin/passwd` contient le code (binaire) de la commande `passwd`, il est propriété de l'utilisateur `root` et tout utilisateur est autorisé à l'exécuter.

L'utilisateur `Gaston` lance la commande `passwd` : il initialise un processus exécutant le code contenu dans `/usr/bin/passwd`. Le set-uid bit ayant été positionné sur celui-ci, les droits de ce processus



Le système Unix

30/09/2003

seront ceux du propriétaire de ce fichier, cad ceux de *root*. C'est ainsi que l'utilisateur *Gaston* peut modifier un fichier (ici, */etc/passwd*) sur lequel il n'a pas droit d'écriture.



3 - Les principales commandes du système

3.1 - Deux type de commandes

3.1.1 - Les commandes externes

Elles correspondent à des fichiers exécutables. Elles entraînent la création d'un nouveau processus. Elles sont appelables dans n'importe quel langage de commande.

3.1.2 - Les commandes internes

Particulière au langage de commande.

3.2 - Code de retour d'une commande (*exit status*)

Toute commande Unix renvoie un entier qui renseigne sur la façon dont s'est déroulée l'exécution, avec le principe **ZERO = TOUT C'EST BIEN PASSE**.

Ce code de retour est stocké dans la variable d'environnement `$?` .

3.3 - Fichiers standards d'E/S

3.3.1 - Définition

Au login d'un utilisateur, un shell est lancé pour le compte de l'utilisateur. Le processus exécutant ce shell ouvre pour défaut trois fichiers:

- ❑ **l'entrée standard**, de descripteur 0, appelée *stdin*, qui, par défaut est le clavier
- ❑ **la sortie standard**, de descripteur 1, appelée *stdout*, qui, par défaut est l'écran
- ❑ **la sortie erreur standard**, de descripteur 2, appelée *stderr*, qui, par défaut est l'écran

3.3.2 - Les redirections

Au niveau du shell, on peut rediriger les fichiers d'E/S standards de façon très simple pour une commande donnée : mais attention, le processus l'exécutant n'aura aucune idée des redirections gérées par le shell.

Voici les différentes manières de rediriger les sorties et les entrées d'une commande, sachant qu'on peut en combiner plusieurs sur une commande donnée :

- `digit>name` : le descripteur de numéro `digit` est associé avec le fichier physique de nom `name` ouvert en écriture. Si le fichier n'existait pas auparavant, il est créé. Si `digit` est omis, la valeur 1 est prise par défaut (sortie standard).
- `digit>>name` : la même chose, mais le fichier est ouvert en mode ajout (on écrit à la fin du fichier existant).
- `digit<name` : la même chose, mais cette fois le fichier `name` est ouvert en lecture ; il doit exister. Si `digit` est omis, la valeur 0 est prise par défaut (entrée standard).
- `digit<<word` : associe le descripteur `digit` avec le fichier du programme en cours, à partir de la ligne qui suit cette déclaration jusqu'à la ligne précédant celle débutant par le terme `word`. Cela



Le système Unix

30/09/2003

permet d'inclure dans des programmes des zones de texte à utiliser ou à recopier dans d'autres fichiers au moment de l'exécution. La commande redirigée de cette manière utilise comme entrée standard les lignes qui suivent immédiatement dans le programme, jusqu'au terme `word`.

- `d1>&d2` : le descripteur `d1` est associé en mode écriture avec le fichier déjà associé au descripteur `d2`. Si `d1` est omis, la valeur 1 est prise par défaut ; `d2` ne peut pas être omis. Le fichier déjà associé au descripteur `d2` doit avoir été ouvert lui aussi en écriture.

- `d1>>&d2` : même chose mais `d1` est associé en mode ajout.

- `d1<&d2` : même chose, mais `d1` est associé en mode lecture ; si `d1` est omis, la valeur 0 est prise par défaut ; le fichier associé à `d2` doit avoir été ouvert en lecture.

Les redirections ci-dessus sont placées à la fin d'une commande et ne prennent effet que sur la commande qui précède juste. Pour modifier une fois pour toutes la redirection d'un descripteur à l'intérieur d'un programme shell, on utilise la syntaxe suivante :

exec **redirection**

la redirection **redirection** qui prend une des formes vues ci-dessus est appliquée au shell en cours et donc par héritage, à toutes les commandes que celui-ci lancera.

Si vous tapez par exemple `exec >/tmp/toto`, vous n'aurez plus sur votre console que le prompt et les messages d'erreur qui vont s'afficher ; le résultat de toutes vos commandes (comme `ls`, `pwd`, etc...) sera envoyé dans le fichier `/tmp/toto`.

Evidemment, c'est peu intéressant de travailler de cette manière, par contre, on peut vouloir rediriger tous les messages d'erreur des commandes d'un programme vers un fichier particulier : dans ce cas, il suffit de placer en début de programme une commande du type :

exec 2>liste_erreurs

Il existe d'autres types de combinaison, mais qui sont bien moins usités. Par exemple, il est possible de faire une redirection en mode entrée-sortie (`digit<> name`), mais le mode sortie est en fait un mode ajout, ce qui n'est pas vraiment intéressant. De toute façon, la quasi-totalité des commandes UNIX n'acceptent pas d'avoir des descripteurs d'entrée et de sortie qui pointent sur un même fichier : on sera presque toujours obligé d'utiliser un fichier intermédiaire pour simuler le mode "entrée-sortie" sur un fichier ouvert. Un bon usage des redirections ci-dessus permet déjà beaucoup de manipulations d'entrées-sorties.

Exemple :

```
$ grep truc <toto 2>trace
```

Le shell qui évalue la ligne interprète :

`<toto` par : ouvrir le fichier `toto` en lecture et l'affecter au descripteur No 0 (entrée standard) du nouveau processus `grep`

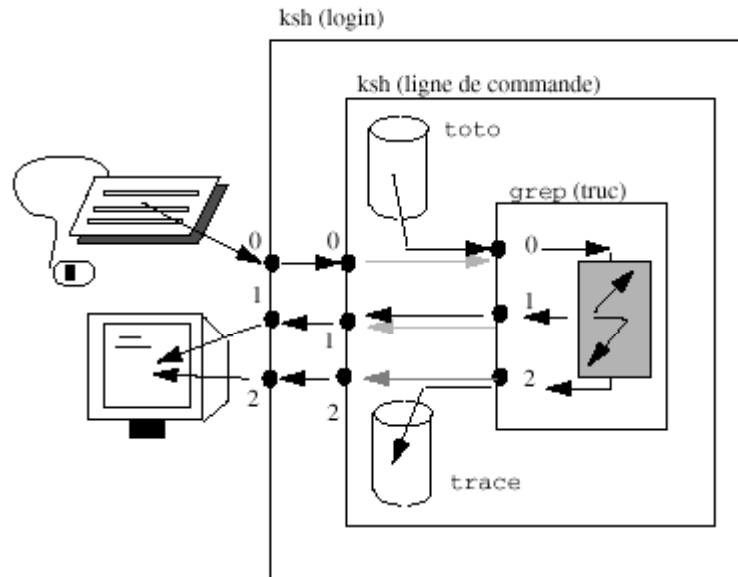
`2>trace` par : ouvrir le fichier `trace` en écriture et lui affecter le descripteur 2.

Comme aucune directive ne touche le descripteur 1, on ne le modifie pas. L'ancêtre du shell actuel, le shell de login par exemple, l'avait déjà affecté à la console : il y est encore par héritage. Le nouveau processus `grep` lit les lignes présentes sur son entrée standard (descripteur 0), et recopie celles contenant la chaîne "truc" sur la sortie standard (descripteur 1).

Le système Unix

30/09/2003

Résultat : les lignes du fichier toto qui contiennent la chaîne “truc” s’affichent à l’écran, et le fichier trace est vide. On peut suivre les connexions des différentes sorties de la commande sur le schéma suivant :



flèches grisées : l'état des connexions de la commande “grep” sans les redirections ;
 flèches noires : avec les redirections.

3.4 - Les filtres

On appelle filtre un programme qui lit des données sur l'entrée standard (stdin) et qui écrit ses résultats sur la sortie standard (stdout) après les avoir filtrées. Les commandes les plus courantes en la matière sont :

- cat pour concaténer des fichiers
- grep pour rechercher une chaîne de caractères dans un texte
- sort pour trier
- wc pour compter les mots (word count), cela inclut les lignes, mots et caractères
- wc -l nom_fichier pour compter le nombre de lignes dans un fichier
- wc -w nom_fichier pour compter le nombre de mots dans un fichier
- wc -c nom_fichier pour compter le nombre de caractères dans un fichier.

3.5 - Les tubes

L'opérateur | permet de lancer les deux commandes contiguës en reliant la sortie standard (descripteur 1) de la première commande dans l'entrée standard (descripteur 0) de la seconde. Le nom “pipe” donne l'idée d'un tuyau par lequel les données transitent d'une commande à l'autre.

Il est possible ainsi de chaîner autant de commandes que l'on désire, le seul impératif étant que chacune des commandes lise les données d'entrée sur le descripteur 0 et fournisse les résultats sur le descripteur 1 (éventuellement après une redirection).



Le système Unix

30/09/2003

Les deux ou N commandes reliées par des “pipes” sont démarrées simultanément dans autant de processus différents et disparaissent une à une lorsque leur entrée standard est vide. Le shell passe à l’exécution de la commande ou de la ligne de programme suivante lorsque toutes les commandes liées par des pipes ont disparu.

Exemple :

```
$ ls
toto1.c toto2.c toto_base.c tutu
$ ls | grep toto | sort
toto1.c
toto2.c
toto_base.c
```

L’exemple ci-dessus est bien compliqué pour le résultat escompté, mais il montre comment il est possible de combiner plusieurs commandes UNIX pour en constituer une plus puissante ou mieux adaptée au besoin courant.

3.6 - D'autres commandes Unix

3.6.1 - Identification par le système

id : permet d’obtenir des informations relatives à un utilisateur. Sur System V la commande **id** limite l’affichage des groupes à celui du groupe principal.

logname : fournit la valeur de la variable d’environnement LOGNAME, qui est initialisée lors de la connexion.

newgrp : permet à l’utilisateur de changer de groupe et d’utiliser un nouveau groupe dont il est membre. Cette commande a pour effet de changer son GID pour celui du nouveau groupe choisi. Cette commande sans argument provoque le retour au groupe principal.

su : permet de prendre l’identité d’un autre utilisateur.

3.6.2 - Manipulation de fichiers et espace disque.

cat fichier(s)

Ecrit contenu de(s) *fichier(s)* sur la sortie standard

du {répertoire(s)}

Affiche sur sortie standard l'espace-disque utilisé par toute l'arborescence courante {ou par celle(s) spécifiée(s)}

df -k {file_system ou répertoire}

Affiche sur sortie standard l'espace-disque libre et utilisé par tous les utilisateurs sur tous les file systems montés {ou sur celui spécifié}

file fichier(s)

Identifie le type des *fichier(s)* en fonction du contenu.

find répertoire(s) condition(s)

Recherche dans toute l'arborescence de chaque *répertoire* spécifié les fichiers satisfaisant au(x) *condition(s)* donnée(s).

En faisant précéder la condition de \! on recherche les fichiers ne satisfaisant pas la condition donnée.

Quelques **conditions** possibles (plusieurs critères peuvent être combinés) :

-name '*fichiers*' recherche *fichiers* de noms spécifiés



Le système Unix

30/09/2003

-user <i>username</i>	fichiers appartenant à utilisateur <i>username</i>
-type <i>type</i>	fichiers de <i>type</i> donné : d= directory, f= fichier, l= lien symbolique...
-mtime <i>n</i>	fichiers modifiés depuis <i>n</i> jours
-perm <i>nnn</i>	fichiers dont la protection est <i>nnn</i>
-print =	affichage des résultats avec leur path complet
-exec <i>commande</i>	applique <i>commande</i> sur tous les fichiers satisfaisants

more { *fichier(s)* }

Affichage page par page du contenu du (des) *fichier(s)* texte spécifiés.

quota -v

Affiche l'usage de l'espace-disque par l'utilisateur (maximum autorisé et espace utilisé) sur chaque file-system

who

Affiche la liste des utilisateurs connectés sur la machine courante.

cmp

permet de comparer deux fichiers de nature quelconque (ascii ou binaire) Sa syntaxe est :
cmp [-l] [-s] file1 file2

diff

affiche les différences entre deux fichiers ascii. Elle est capable de se resynchroniser sur les séquences de lignes identiques. L'option **-b** permet d'ignorer les espaces traînants qui se trouvent après les derniers caractères visibles en fin de ligne.

split

permet de scinder un fichier en fichiers plus petits. Le fichier passé en argument est alors découpé en fichiers de 1000 lignes appelés respectivement par le nom passé en argument suivi des caractères aa, ab, ac, ...

tr

est un filtre permettant d'appliquer un transcodage des caractères entre l'entrée et la sortie standard. Exemple : tr a-z A-Z

tar

permet de sauvegarder une arborescence de fichiers. Elle fonctionne d'une manière qui est décorrélée de tout système de fichiers. On peut donc sauvegarder des parties de systèmes de fichiers différents sur une même bande magnétique.

Exemple : tar -c /usr

Ceci correspond à la sauvegarde complète du répertoire /usr sur le lecteur de bande. Si on désire indiquer le fichier spécial de façon explicite on utilise l'option f. L'option v (verbiage) affiche les noms des fichiers qui sont sauvegardés sur la sortie des erreurs.

Exemple : tar cfv archive.tar usr

Ceci va créer une archive archive.tar qui va contenir le repertoire usr.

Pour extraire les fichiers on utilise l'option x a la place de l'option c.

3.6.3 - Commandes d'impression

lp {-d imprimante} {*fichier(s)*}

Insère l'entrée standard ou le(s) *fichier(s)* sur la file d'attente de l'*imprimante* spécifiée. Si aucune *imprimante* n'est désignée, utilise celle définie par l'utilisateur par la variable d'environnement LPDEST (ou l'imprimante par défaut définie par l'administrateur)

lpstat {*imprimante*}

Indique *Id* (sous la forme *imprimante-No*) et statut des requêtes d'impression en attente sur toutes les imprimantes (ou l'imprimante spécifiée).

cancel *Id(s)* , **cancel -u** *username*

Avorte la(les) job(s) d'impression spécifié(s) par *Id(s)*,
ou avorte tous les jobs d'impression soumis par l'utilisateur *username*



4 - La gestion des processus

4.1 - Notion de processus

On appelle **processus** l'exécution d'un programme à un moment donné. Un processus va donc naître, vivre et mourir.

Un **programme**, lui, est un objet inerte qui est stocké sur un disque dur dans un fichier ordinaire dit « binaire exécutable ».

4.2 - Types de processus

4.2.1 - Processus "utilisateur"

Le fait de se connecter au serveur Unix initie une session dont le processus maître est le processus shell créé : cette session et donc les processus lancés successivement depuis ce shell seront « attachés » au terminal de l'utilisateur.

Les processus « utilisateur » sont peu prioritaires.

4.2.2 - Processus "système"

Certains de ces processus sont lancés au boot du serveur et vivront jusqu'à son arrêt : c'est le cas du processus *init* qui est chargé de lancer les procédures de connexion sur tous les terminaux reliés en liaison série.

Certains de ces processus, appelés **démons**, sont lancés en arrière-plan et détachés de tout terminal et assurent l'accès à des services (ex : *lpsched* gestionnaire d'impression).

Les processus « système » sont très prioritaires.

4.3 - L'ordonnanceur de tâches

Le système gère une table des processus dans laquelle chaque processus possède une entrée allouée à la création du processus ; cette entrée contient toutes les informations nécessaires au système lorsque le processus n'est pas actif.

Chacun leur tour, les processus inscrits dans cette table sont chargés en mémoire et bénéficient de toutes les ressources de l'ordinateur pendant un certain temps puis leur image mémoire est sauvegardée en zone de swap. Ce va-et-vient est réalisé par le démon appelé ordonnanceur de tâches.

Image mémoire d'un processus :

- Segment de texte (code)
- Segment de données utilisateur (statique/dynamique)
- Segment de données système (gestion des E/S)
- Valeur des registres



Le système Unix

30/09/2003

4.4 - Caractéristiques d'un processus

4.4.1 - Bloc de contrôle

Chaque processus possède un bloc de contrôle (BCP) identifié par un numéro appelé **PID** (Processus Identifier). Dans ce BCP, on trouve toutes les caractéristiques du processus et en particulier :

- ❑ **UID** propriétaire réel
- ❑ **EUID** propriétaire effectif
- ❑ **TTY** terminal de contrôle
- ❑ **PRI** priorité calculée dynamiquement par l'ordonnanceur de tâches

4.4.2 - Calcul de la priorité d'un processus

Règle 1 : Les processus système sont plus prioritaires que les processus utilisateur

Règle 2 : Plus un processus attend dans la table des processus, plus il devient prioritaire.

Remarque : Cette stratégie d'allocation des ressources de l'ordinateur ne permet pas de gérer le temps réel : elle est adaptée à une stratégie « temps partagé ».

4.4.3 - La commande nice

Elle permet à tout utilisateur d'essayer de diminuer la priorité de ses processus et seulement à l'utilisateur « root » d'essayer d'augmenter celle de ses processus.

4.5 - Affichage des caractéristiques d'un processus

La commande *ps* (process status) avec l'option *l* affiche les caractéristiques essentielles des processus de l'utilisateur en cours.

Les autres options les plus utilisées sont :

- ❑ Option *e* tous les processus du serveur unix
- ❑ Option *t* *terminal* tous les processus attachés à ce tty
- ❑ Option *u* *utilisateur* tous les processus appartenant à cet utilisateur

Exemple

```
$ps -l
F S UID PID PPID PRI ... TTY ... CMD
1 S 310 137 1 30 ... tty23 ... ksh
1 R 310 203 137 70 ... tty23 ... ps
```

A partir des données affichées, il est possible de reconstituer l'arborescence des processus qui sont attachés au terminal */dev/tty23* et qui appartiennent à l'utilisateur dont l'UID est 310.

La colonne **PRI** montre que le processus **ksh** est davantage prioritaire que le processus **ps** : petit **PRI** → grande priorité .

La colonne **S** montre l'état des processus : **T** pour stoppé, **Z** pour terminé (zombie), **W** pour attente, **S** pour sommeil en attente d'événement et **R** pour en cours d'exécution.

Les données affichées par **ps** sont déjà périmées au moment où on les lit !



Le système Unix

30/09/2003

4.6 - Processus en avant plan

Un processus qui s'exécute en **avant-plan** monopolise complètement son terminal d'attachement et peut être contrôlé au clavier par la frappe de certains caractères provoquant l'envoi de **signaux** meurtriers au processus :

- `<intr>` envoi du signal **SIGINT** qui cause en principe la mort du processus
- `<quit>` envoi du signal **SIGQUIT** qui cause en principe la mort du processus avec création d'un fichier **core** contenant l'image mémoire du processus

Exemple

```
$sleep 1000
$
```

Le shell lance la commande et attend qu'elle soit achevée pour afficher de nouveau son invite : il s'agit du **mode synchrone**. Si l'utilisateur frappe pendant ce temps sur le caractère `<intr>`, il tue le processus *sleep* et l'invite du shell est réaffichée.

Remarques

- Les caractères `<intr>` et `<quit>` ne sont pas standardisés. Aussi, on utilise la commande *stty -a* pour retrouver leur valeur ainsi que toutes les caractéristiques du terminal.
- Le comportement d'un processus à la réception d'un signal peut être programmé. Aussi il ne faut pas s'étonner si certains processus ignorent les signaux **SIGINT** et **SIGQUIT**.
- La commande *kill -INT <PID>* lancée sur un autre terminal aura le même effet sur le processus `<PID>` que la frappe du caractère `<intr>` sur le clavier du terminal de contrôle de ce processus.

4.7 - Processus en arrière plan

Un processus qui s'exécute en **arrière-plan** ne monopolise plus son terminal d'attachement mais ne peut pas être contrôlé au clavier.

Exemple:

```
$ sleep 1000 &
[1] 4125
$
```

Le "&" signifie que l'on a lancé la commande *sleep* en arrière-plan et 4125 est le PID de cette commande.

Le shell lance la commande et n'attend pas qu'elle soit achevée pour afficher de nouveau son invite : il s'agit du mode **asynchrone**.

4.8 - Le contrôle de jobs

Sans le contrôle de jobs, un processus vivait toute sa vie soit en avant plan soit en arrière plan. Avec le contrôle de jobs, un processus pourra être changé de plan au cours de son existence.

Lorsqu'une commande (ou groupe de commandes) est lancée en arrière plan, le shell lui associe un numéro de job et inscrit ce job dans une table que l'on peut consulter avec la commande *jobs*.

On peut alors faire passer le job en avant plan avec la commande *fg* (foreground) puis le remettre en arrière plan avec la commande *bg* (background) mais il faut auparavant le stopper sans le tuer : cela est réalisé en tapant au clavier le caractère `<susp>` qui cause l'envoi au job du signal **SIGSTOP**.



Le système Unix

30/09/2003

4.9 - Les signaux asynchrones

4.9.1 - Les signaux

Les signaux disponibles dans un système Unix sont définis dans le fichier `/usr/include/sys/signal.h`. Ils peuvent être affichés avec la commande `kill`. Certains signaux sont utilisés pour tuer les processus.

- SIGHUP 1 fin de session
- SIGINT 2 Interruption <intr>
- SIGQUIT 3 inter.+ core <quit>
- SIGKILL 9 meurtre assuré
- SIGTERM 15 terminaison

D'autres signaux sont utilisés par le contrôle de jobs :

- SIGSTOP 23 demande de suspension pour une reprise ultérieure
- SIGCONT 25 reprise du processus suspendu

Remarques :

Certains processus exécutant des shells-scripts ou des binaires exécutables sont protégés contre certains signaux ou traitent ces signaux de façon non standard.

Le signal SIGKILL est l'arme absolue contre un processus, car il ne peut être ignoré ni même traité par une fonction de déroutement.

4.9.2 - La commande kill

La commande `kill` permet de tuer un processus en lui envoyant le signal SIGTERM, signal envoyé par défaut, ou tout autre signal. Sa syntaxe est :

```
kill [-signo] pid ...
```

`signo` désigne le numéro du signal. Il peut être exprimé sous la forme d'un numéro ou d'une chaîne de caractères. L'option `-l` liste les signaux disponibles.

Exemple :

```
$ kill 24220
```

```
$ kill -15 24220
```

```
$ kill -TERM 24220
```

Ces trois exemples sont totalement identiques, le signal 15 étant celui par défaut.



5 - L'éditeur de texte VI

5.1 - Etude rapide

Vi est l'éditeur de texte standard d'UNIX, il est plein-écran et par conséquent ne fonctionne correctement que si la variable d'environnement TERM est correctement positionnée.

5.1.1 - variable d'environnement TERM

Vérifier d'abord que cette variable est bien définie comme variable d'environnement en utilisant la commande env.

La valeur de cette variable doit être vt220 sur les ordinateurs émulés en terminaux ou en telnet. Modifier éventuellement la valeur de la variable TERM.

Sur les ordinateurs et certains serveurs Unix, une mappe en entrées et une mappe en sortie doivent être mises en place par vos soins avec la commande suivante : setmaps -t vt220

Attention ces modifications devront être pratiquées à chaque connexion, sur la ligne de commande ou mieux dans votre fichier de configuration privée.

5.1.2 - Entrée et sortie de vi

Faire une copie d'un fichier (par exemple /etc/profile) dans votre catalogue principal sous le nom de kop pour expérimenter l'éditeur vi.

Lancer alors la commande vi avec comme argument le nom de ce fichier. vi affiche alors la première page de copie et des tildes à la place des lignes vides si nécessaires.

Le texte est stocké dans un buffer temporaire; les opérations d'édition modifient le buffer, pas le fichier original.

Pour sauver le fichier à un moment donné on peut taper ": w" suivi de return.

Pour sauver l'édition et quitter vi, presser la touche "shift" tout en appuyant deux fois sur la touche Z ou alors ": wq" suivi de "return".

Pour quitter vi sans sauvegarde, taper ": q !" suivi de "return".

5.1.3 - Le mode commande

Editer de nouveau le fichier copie : "vi kop", vi se met initialement en mode commande.

Les touches invoquent alors des fonctions d'éditor, applicables immédiatement (il y a plus de cent fonctions !!!)

5.1.3.1 - Déplacement du curseur

Les flèches... ou

"CTRL U" déroulement d'un demi-écran vers le haut

"CTRL D" déroulement d'un demi-écran vers le bas

"CTRL F" page suivante

"CTRL B" page précédente

"?blabla" : recherche de la chaîne blabla du curseur vers le haut

"/blabla" : recherche de la chaîne blabla du curseur vers le bas



Le système Unix

30/09/2003

"n" : passage à l'occurrence suivante de la chaîne.

5.1.3.2 - Destruction

"x" : détruit un caractère

"dd" : détruit une ligne

"5dd" : détruit cinq lignes

5.1.3.3 - Refaire/ défaire

"u" : défait la dernière modification (dest. ou insert).

"." : refait la dernière modification (dest. ou insert).

5.1.3.4 - Rafraîchissement de l'écran

"CTRL L" réaffichage de l'écran, utile quand un utilisateur quelque peu facétieux vous a expédié un message par write.

5.1.3.5 - Modification

"r <car>" substitue au caractère pointé le caractère donné

"R <chaîne> <ESC>" substitue au caractère pointé la chaîne donnée

"J" joint la ligne courante à la suivante.

5.1.3.6 - Buffer standard

"5yy" : Copie les cinq lignes qui suivent le curseur dans le buffer

"p" : Copie le contenu du buffer en dessous du curseur

"P" : Copie le contenu du buffer au dessus du curseur.

5.1.4 - Le mode insertion

Sous vi le curseur représente le caractère qu'il recouvre.

Des caractères peuvent être insérés :

"i" : avant le curseur

"a" : après le curseur

"I" : au début de la ligne

"A" : à la fin de la ligne.

En mode insertion, "BACKSPACE" détruit les caractères insérés, mais uniquement sur la ligne courante; les caractères détruits restent visibles jusqu'à ce que "ESC" soit tapé.

Pour quitter le mode insertion taper "ESC"

5.2 - Configuration de vi

5.2.1 - En mode commande

On s'assure d'être bien en mode commande en tapant <ESC> et on peut alors configurer vi en positionnant ou non certaines options.

":set" permet de connaître les options positionnées

":set all" permet de connaître toutes les options disponibles.



Le système Unix

30/09/2003

Pour positionner par exemple l'option de numérotation des lignes: ": set number"
pour inhiber cette même option: ": set nonumber"

Quelques options fortement conseillées :

- redraw : meilleure gestion de l'écran (indispensable)
- showmode : montre le mode (insertion/commande)

Quelques options bien pratiques pour rédiger un programme en C :

- autoindent : indentation automatique ("CTRL D" pour l'éviter)
- showmatch : montre les associations de parenthèses/accolades
- number : numérote les lignes à l'affichage

5.2.2 - A la connexion

Au lieu de configurer vi en mode commande à chaque fois qu'il est lancé, il peut être beaucoup plus pratique de le configurer une fois pour toute et automatiquement, dès que l'utilisateur se connecte. En effet quand un utilisateur se connecte au système, un shell est lancé pour lui et ce shell exécute un script-shell de profil du catalogue principal de l'utilisateur.

Il suffit donc d'ajouter une ligne à ce fichier pour donner à la variable d'environnement EXINIT une valeur correcte qui sera utilisée pour configurer vi à chaque invocation de cet éditeur de texte.

Exemple :

```
EXINIT = "set redraw showmode"  
export EXINIT
```

5.2.3 - A chaque invocation

De façon encore plus pratique on peut configurer vi à chaque invocation en créant un fichier .exrc qui contiendra les options choisies.

On peut même créer un fichier .exrc dans chacun de ses répertoires avec des configurations différentes de vi.

Si un utilisateur a créé dans son catalogue principal deux catalogues, l'un nommé sources où il stocke ses fichiers sources en C et l'autre scripts où il stocke ses scripts en shell, il utilisera deux fichiers .exrc. Exemple :

```
$cat $HOME/ sources/.exrc  
set redraw showmode autoindent number showmatch  
$cat $HOME/ scripts/.exrc  
set redraw showmode autoindent  
$
```

5.3 - Avantages et inconvénients

vi est ancien, des problèmes d'affichage et la distinction entre mode commande et mode insertion le rendent plus délicat à apprendre que des éditeurs récents beaucoup plus conviviaux.

Mais vi a des avantages, il est :

- standard : présent sur tous les systèmes UNIX
- portables : fonctionne sur tous les écrans
- rapide : commande très efficaces pour les utilisateurs expérimentés;

Il existe par ailleurs d'autres éditeurs de textes sous unix:

ed proche du edlin de DOS, un vrai régal!!!



Le système Unix

30/09/2003

EMACS plein écran multi-fenêtrage, langage d'extension basé sur LISP, mode spécial pour programmer en C, LISP... mais gros consommateur de ressources

5.4 - Compléments

5.4.1 - Appel d'une commande UNIX

On peut sans quitter vi lancer une commande UNIX : "ESC" peut être sûr d'être en mode commande puis ": ! cmd" où cmd est n'importe quelle commande UNIX

5.4.2 - Utiliser des buffers nommés

"ESC" pour être sûr d'être en mode commande
"a4YY extrait quatre lignes dans le buffer a
"ap met le contenu du buffer a après le curseur
"aP met le contenu du buffer a avant le curseur

5.4.3 - Filtrer du texte

Ceci est vraiment TRES PRATIQUE !!! Des lignes de texte peuvent être passées comme entrée standard d'un filtre UNIX quelconque; les lignes choisies seront alors remplacées par les sorties standards du filtre choisi.

Par exemple pour avoir la date dans un fichier :

- taper date sur une ligne du fichier
- quitter le mode insertion en tapant "ESC"
- placer le curseur sur la ligne de date
- filtrer la ligne avec bash en tapant "!!bash"

On peut ici appeler n'importe quelle commande et pas seulement bash.



6 - L'interpréteur de commande bash

6.1 - Qu'est-ce qu'un shell ?

Le rôle principal d'un shell est de lancer l'exécution des commandes tapées par l'utilisateur. c'est aussi le shell qui gère l'environnement de travail de l'utilisateur (alias, historique, etc.).

Les shell les plus connus sont le Bourne-Shell *bsh*, Korn-Shell *ksh*, le C-Shell *csh* et le Bourne-Again-Shell *bash*.

Un shell est un processus qui fonctionne en boucle perpétuelle:

1. Affiche d'une invite (le contenu de la variable d'environnement *PS1*; par défaut : \$)
2. Lecture de la ligne de commande et réalisation de substitutions diverses
3. Lancement d'un ou plusieurs processus pour exécuter cette ligne de commande
4. Retour en 1 après la fin de cette exécution(avant-plan) ou immédiatement (arrière-plan)

6.2 - Configuration du shell bash

Des fichiers de configuration existent pour tous les shell. Ces fichiers permettent de les configurer automatiquement au début d'une session de travail (ou au lancement d'un nouveau shell à partir de la ligne de commande)

Ces fichiers sont, dans l'ordre :

- ❑ */etc/profile* lors de l'ouverture de la session (configuration collective).
- ❑ *\$HOME/.bash_profile*, ou *\$HOME/.profile* (si aucun *.bash_profile* n'est présent) lors de l'ouverture de la session (configuration privée).
- ❑ *\$ENV* toujours (configuration privée).

A la fin de la session de travail, le fichier *\$HOME/.bash_logout* est exécuté.

6.3 - Fichier de commandes (script-shell)

Un **script-shell** est un ensemble de commandes reconnues par le shell stockées dans un fichier.

Quatre possibilités pour exécuter un script-shell :

- ❑ Taper directement le nom du script sous l'interpréteur de commande. L'interprétation s'effectue dans un sous-shell. Il faut néanmoins que l'utilisateur ait impérativement le droit d'exécution sur le fichier.
- ❑ Taper directement le nom du script sous l'interpréteur de commande en le faisant précéder d'un point. L'interprétation s'effectue dans le shell courant
- ❑ Utiliser la commande *exec* suivie du nom du script. L'interprétation s'effectue dans un sous-shell non interactif.
- ❑ Utiliser la commande *bash* suivie du nom du script. L'interprétation s'effectue dans un sous-bash.

Les **paramètres positionnels** sont des variables shell qui permettent l'accès aux arguments d'un shell-script :

\$0 nom du script \$1,...,\$9 arguments (du 1^{er} au 9^{ème})
\$# nombre d'arguments \$* liste de tous les arguments

Exemple de script-shell :

```
$cat pollux
```



Le système Unix

30/09/2003

```
echo Nom du programme : $0
echo $1 $2
echo Fin du programme
$ pollux margotte azalee
Nom du programme : pollux
Margotte azalee
Fin du programme
$
```

6.4 - Les variables du shell

6.4.1 - Variables locales

Une variable ne peut contenir qu'une chaîne de caractères :

```
$ val=berliet
$ echo $val
berliet
$ unset val
$ echo $val
```

\$

Seul le shell courant a connaissance de ces variables ; la liste des variables du shell est donnée par la commande *set*.

6.4.2 - Variables d'environnement (variables exportées)

Certaines variables sont prédéfinies au niveau du shell et sont lisibles par la descendance de ce processus shell (i.e. les sous shell lancés à partir de ce shell).

ENV	Nom du fichier de configuration
HOME	Répertoire de connexion.
MAIL	Chemin indiquant le répertoire du courrier
MAILCHECK	Intervalle en sec au bout duquel le mail est contrôlé
PS1	Invite principale du shell en mode interpréteur
PS2	Invite secondaire du shell en mode programmation
PS3	Valeur de l'invite de sélection dans une boucle <i>select</i>
PS4	Invite de trace
PATH	Chemin de recherche pour l'exécution des commandes
PPID	Numéro du processus père du shell.
PWD	Répertoire de travail courant défini par la commande <i>cd</i>
TERM	Nom du type de terminal.
SHELL	Indique le shell de login.

La commande *export* permet d'ajouter des variables exportées à l'environnement

La commande *env* permet de visualiser les variables d'environnement.

6.4.3 - Typer une variable

La commande *typeset* permet de typer une variable.

Les options sont :

-f : fonction	-l : Majuscules vers minuscules	-i : entier
-r : accessible en lecture	-u : minuscules vers majuscules	-x : exporté



Le système Unix

30/09/2003

6.4.4 - Quotage des variables

Les chaînes de caractères peuvent être délimitées par trois caractères :

- ❑ Simple quote '...': les caractères inclus entre 2 simples quotes ne sont pas évalués, ils conservent leur valeur littérale.
- ❑ L'anti quote `...`: les caractères inclus entre 2 anti quotes sont évalués, et interprétés.
- ❑ Double quote "...": les caractères inclus entre 2 doubles quotes ne sont pas évalués et conservent leur valeur littérale à l'exception de \$ ` et \.

6.4.5 - Variables de substitution

- ❑ \$! Numéro du dernier processus en arrière plan
- ❑ \$? Code retourné par la dernière commande
- ❑ \$\$ PID du shell courant

6.5 - Les alias

Ce mécanisme permet de donner des surnoms ou des abréviations à des commandes Unix.

La commande *alias* sans argument donne la liste des alias.

La commande *alias nom=valeur* initialise un alias

La commande *unalias* supprime un alias.

6.6 - Masque des droits d'accès

Cette fonctionnalité permet de protéger automatiquement les fichiers à leur création en inhibant certains droits. Le masque courant est donné par la commande *umask* ; il est modifié par la commande *umask xyz*, où *xyz* est la nouvelle valeur du masque.

On place généralement la commande *umask xyz* dans le fichier *.profile*.

ex. Si on tape *umask 027*, les droits *rwxr-x---* seront positionnés pour les fichiers créés à l'avenir.

6.7 - Traitement des interruptions

La commande interne *trap* permet à un processus de spécifier quel type de comportement il souhaite avoir lorsqu'un signal particulier lui est adressé.

Syntaxe : *trap [argument] liste de signaux*

Si aucun argument n'est mis, le comportement standard est rétabli.

Exemple : *trap '' 2 3* Les signaux SIGQUIT et SIGINT sont ignorés

Si *argument* est une commande ou une série de commandes ou une fonction, elles seront exécutées à la réception du signal.

Exemple : *trap 'ls -ail' 2* A chaque réception de SIGINT, on exécute un listing du répertoire.

trap sans argument donne la liste des captages déjà défini.

6.8 - Les instructions du shell

6.8.1 - Commandes de test

Syntaxe : **test** *expr* ou [*expr*] ou [[*exp*]]

Remarque : Attention il faut un espace après [et avant]



Le système Unix

30/09/2003

```
if [ -r beurk ]
then echo "beurk est lisible"
else echo "beurk n est pas lisible"
fi
```

6.8.2.2 - Boucle itérative POUR / FINPOUR

```
for variable in liste
do
commandes
done
```

- Si la liste est omise, *variable* prend successivement par positionnement les paramètres passés au script.
- Si la liste est remplacée par *, *variable* prend successivement les références existantes dans le répertoire courant.

Exemple 1

```
$ cat goldorak1
for i in un deux trois quatre
do
  echo $i
done
$ goldorak1
un
deux
trois
quatre
```

Exemple 2

```
$ cat goldorak2
for i
do
  echo "Bonjour $i"
done
$ goldorak2 alcor venusia
Bonjour alcor
Bonjour venusia
```

Exemple 2

```
$ cat goldorak3
for i in *
do
  echo $i
done
$ goldorak3
titi
tata
toto
```

en supposant que ces trois fichiers sont dans le répertoire courant.

6.8.2.3 - Boucle conditionnelle TANTQUE et REPETER JUSQU'A

```
while test
do
commandes
done
until test
do
commandes
done
```

Sortie de boucle : break

Exemple

```
$cat essailwhile
```



Le système Unix

30/09/2003

```
#Essai de la boucle while
while [ $1 != fin ]
do
    echo $1
    shift
done
$essailwhile 1 titi 5 fin toto
1
titi
5
```

6.8.2.4 - Structure de choix CASE

```
case chaine in
    motif1[motif2] ... ) liste_commandes ;;
    ...
esac
```

Le caractère * est généralement mis en dernier du motif pour exprimer le cas par défaut.

Exemple

```
case $# in
    0) echo $0 n'a pas d'argument
    1) echo $0 à 1 argument
    2) echo $0 à 2 arguments
    3) echo $0 à 3 arguments
    *) echo $0 à plus de 3 arguments
esac
```