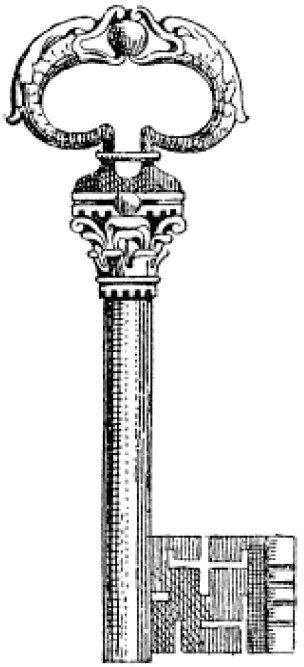
Support de cours



Introduction à l'environnement Unix

Introduction

Unix est le système d'exploitation le plus ancien encore utilisé.
Mais un long chemin a été parcouru depuis sa naissance aux *Bell Laboratories*.

L'histoire du système Unix est indissociable de celle des systèmes d'informations en général. Le langage C, les réseaux TCP/IP, les applications qui sont aujourd'hui le fondement d'Internet (messageries, Usenet, Web, etc.) ont été élaborés en prenant appuis sur Unix.

Unix est une sorte de dinosaure que l'Histoire aurait oublié de supprimer, mais sur lequel l'Évolution continuerait d'exercer ses mutations.

Stable, complexe, versatile, touffu, cohérent, efficace, rugueux, riche, étonnant sont autant d'adjectifs qui permettent de le qualifier.

Déroulement

La durée théorique de ce module de formation est d'une journée.

Il s'agit d'une formation interactive, il est donc tout à fait indiqué d'interrompre le formateur pour lui poser des questions, lui faire préciser certains points, demander l'étude d'un cas particulier, etc.

Historique

Unix est né en 1969 aux *Bell Laboratories* de la main de Ken Thompson.

L'avènement du langage C de Dennis Ritchie a été un élément important pour améliorer sa portabilité.

La première version a été commercialisée en 1978, elle était appelée « V4 ».

Une scission (la plus importante) a lieu :

- les *Bell Laboratories* développent une mouture dénommée « System V » ;
- l'université de Berkeley publie BSD, « *Berkeley Software Distribution* ».

En 1984, la distribution 4.2BSD inclut une couche TCP/IP.

Différents éditeurs sortent leur propre version : XENIX (Microsoft), VENIX (Venturecom), SCO UNIX, etc. Elles sont toutes sont licence AT&T.

Puis des versions indépendantes naissent : le MINIX d'Andrew Tanenbaum, XINU, la distribution GNU de la FSF (*Free Software Foundation*), etc.

Pour palier à ces disparités, les standards XOPEN et POSIX voient le jour.

XWindow (surcouche graphique née du projet MIT Athena) commence a être intégré.

Les Unix d'aujourd'hui

Les systèmes de type Unix constituent une grande famille.

Ces dernières années, un regain de vigueur est observé grâce à ses derniers rejetons (en particulier, Linux et MacOS X).

Les acteurs majeurs qui restent sur le marché sont :

- les Unix des constructeurs :
  Solaris, HP-UX, AIX, MacOS X, etc. ;
- les distributions commerciales basées sur le noyau Linux :
  Suse Novell, Redhat, Mandrake, Yellow Dog Linux, etc. ;
- les Unix libres :
  Debian GNU/Linux, OpenBSD, NetBSD et FreeBSD ;
- les systèmes d'exploitation spécialisés tels que QNX.

Les Unix commerciaux disponibles sur la plateforme Intel (SCO UNIX, BSDI, Digital Unix) sont en déclin.

Caractéristiques

Unix est un système vieux de plus de 30 ans.

Il est disponible sur toutes les plateformes, des micros aux super-calculateurs.

Unix :

- est multi-tâche et multi-utilisateur depuis son origine ;
- propose une interface texte depuis toujours ;
- intègre une interface graphique (XWindow) depuis environ 10 ans;
- est un système riche et cohérent.

Terminaux

L'interface matérielle (réelle ou virtuelle) utilisée pour ouvrir une session est appelée « terminal ». Il peut s'agir :

- de la console physique de la machine ;
- d'un équipement dédié relié directement au système (terminaux passifs) ;
- d'un logiciel d'émulation de terminal utilisant le réseau.

Unix sait gérer beaucoup de terminaux différents grâce aux bases *terminfo* ou *termcap*. Le contrôle du terminal se fait *via* le contenu de la variable d'environnement `$TERM` (nous y reviendrons).

Aujourd'hui, la manière la plus courante d'accéder à un système Unix se fait par l'intermédiaire d'émulateurs de terminaux utilisant le protocole Telnet ou SSH sur TCP/IP.

Beaucoup d'émulateurs sont disponibles : l'application Telnet de Windows, PuTTY, le projet OpenSSH, les émulateurs commerciaux (Reflexion, etc.).

Ouvrir une session

Pour ouvrir une session, il faut :

- un nom de connexion (*login*) ;
- le mot de passe associé ;

Et, si la console du système n'est pas utilisée :

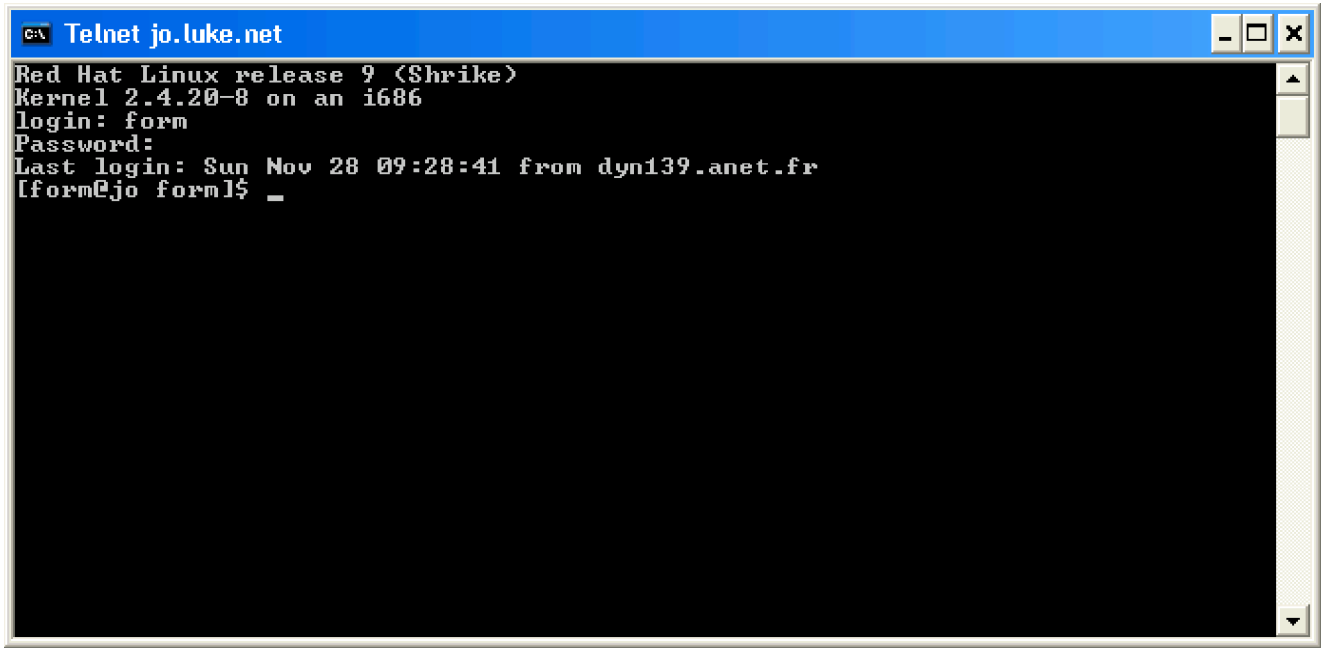
- un émulateur de terminal ;
- le type de protocole utilisé (Telnet ou SSH).

(La casse des caractères du *login* et du mot de passe est significative.)

Ci-contre, un exemple de l'utilisation de l'application Telnet de Windows.

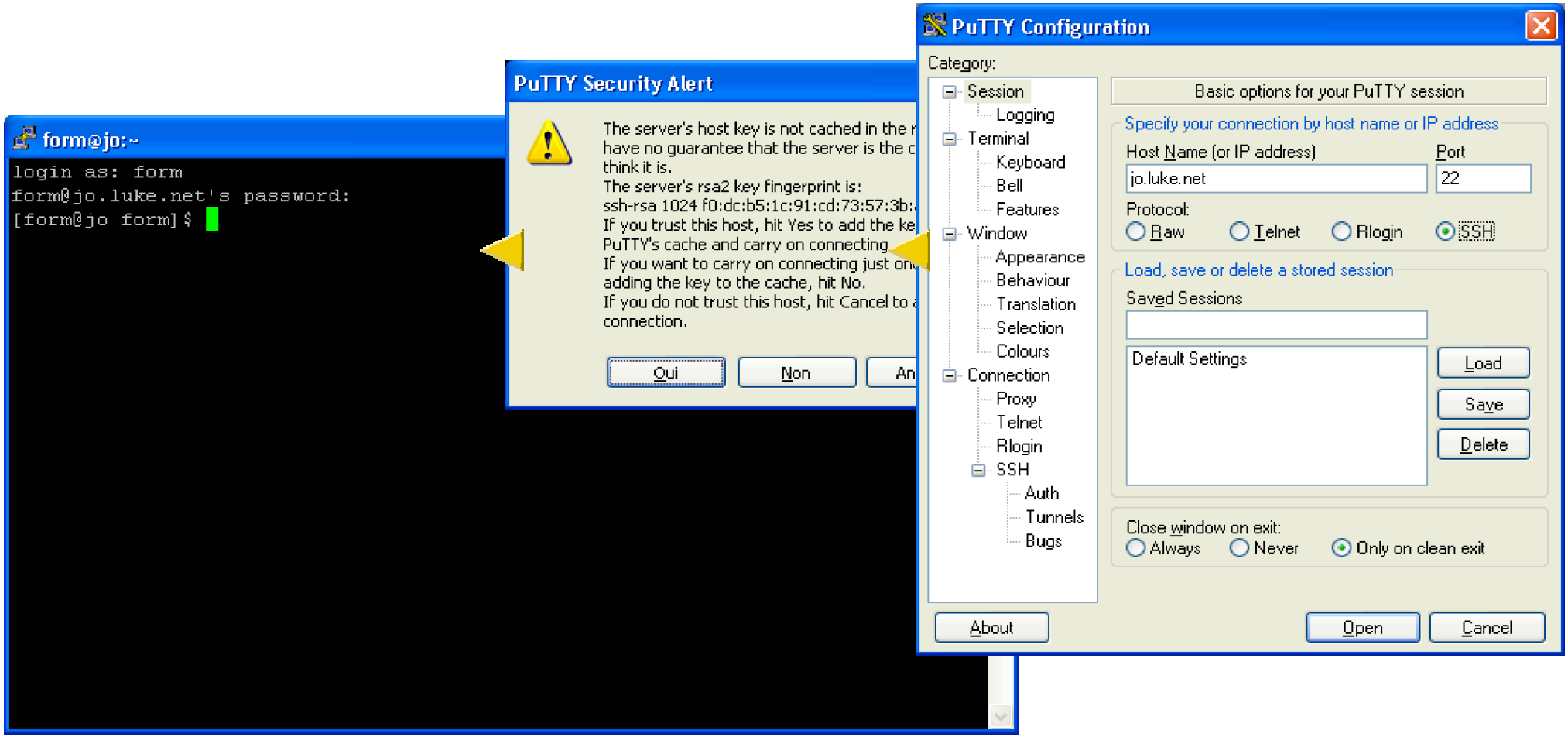
La ligne de commande à utiliser est :

```
c:\> telnet jo.luke.net
```



PuTTY

PuTTY est un émulateur de terminal pour Windows, libre (gratuit), compact, simple et complet. Il supporte les protocoles Telnet et SSH.
Son site est : `http://chiark.greenend.org.uk/~sgtatham/putty/`



Premiers pas

L'invite de commande (*prompt*) de l'interpréteur de commande (*shell*) est affiché. Il peut contenir diverses informations.

Il se termine traditionnellement par :

- le caractère « `$` » pour un utilisateur standard ;
- le caractère « `#` » pour le super-utilisateur (*root*).

Pour mettre fin à la session, on peut utiliser :

- la commande `exit` ;
- ou la commande `logout` ;
- ou la combinaison de touches « Ctrl-d ».

Utilisateurs et groupes

Sous Unix, les utilisateurs sont identifiés par :

- un nom d'utilisateur ;
- un UID (*User ID*) ;
- un GID (*Group ID*) qui représente le groupe primaire de l'utilisateur ;
- une liste de groupes secondaires optionnelle.

La commande `id` permet d'afficher ces informations :

```
$ id
uid=500(form) gid=500(form) groupes=500(form),100(users)
```

L'utilisateur *root* est le super-utilisateur utilisé pour l'administration du système. Par définition, son UID est 0.

```
$ id root
uid=0(root) gid=0(root) groupes=0(root),1(bin),2(daemon),
3(sys),4(adm),6(disk),10(wheel)
```

Unix comparé à DOS/Windows

Voici les différences les plus importantes liées à l'utilisation de la ligne de commande Unix par rapport à celle de DOS/Windows :

- les répertoires sont séparés par « / » et non « \ » ;
- les options des commandes commencent par « - » et non « / » ;
- les noms des fichiers, commandes et variables sont sensibles à la casse ;
- les extensions des fichiers n'ont aucune signification particulière ;
- en général, il n'y a pas de demande de confirmation.

Petite équivalence entre les commandes de base d'Unix et de DOS :

DOS	Unix
`cd`	`cd` et `pwd`
`copy`	`cp`
`move`	`mv`
`dir`	`ls`
`more`	`more`
`edlin`	`ed`

DOS	Unix
`md`	`mkdir`
`rd`	`rmdir`
`type`	`cat`
`del`	`rm`
`deltree`	`rm -rf`
`edit`	`vi`

Le système de fichiers

La commande `pwd` affiche le nom du répertoire courant.

La commande `cd` utilisée sans argument permet de revenir au répertoire par défaut de l'utilisateur connecté.

Les options les utilisées de la commande `ls` sont :

* `-l` : format long ;
* `-a` : affiche les fichiers cachés ;
* `-F` : ajoute un caractère qui précise le type des fichiers.

Par convention, les fichiers et répertoires cachés sont ceux dont le nom commence par un point.

Sous Unix, il n'existe qu'une seule arborescence de fichiers qui commence à la racine « `/` ».

Pour accéder à un périphérique amovible, il faut au préalable l'attacher au système de fichiers avec la commande `mount`. Il sera également nécessaire de réaliser l'opération inverse avec la commande `umount` avant de retirer le média.

Les répertoires standards

Les répertoires standards dans la racine du système de fichiers sont :

`/etc`	configuration du système
`/home`	répertoires des utilisateurs
`/root`	répertoire de l'utilisateur *root*
`/bin /sbin /lib /usr`	programmes et bibliothèques du système
`/opt`	applications tierces
`/var`	données vivantes du système
`/dev`	fichiers spéciaux d'accès au matériel
`/tmp`	répertoire pour les fichiers temporaires

Obtenir de l'aide – `man`

Toutes les commandes et primitives du système Unix sont documentées en détail dans les pages de manuel accessibles avec la commande `man`.

Pour accéder, par exemple, à la documentation de la commande `ls`, il suffit d'exécuter la commande :

```
$ man ls
```

La page de manuel étant affichée, les touches suivantes permettent d'y circuler :

- barre d'espace : avancer d'une page ;
- touche « Entrée » : avancer d'une ligne ;
- « b » remonte d'une page ;
- « G » pour aller à la fin ;
- « p » pour se rendre au début ;
- « / » permet de rechercher une chaîne de caractères ;
- « n » poursuit la recherche ;
- enfin, « q » pour sortir.

Obtenir de l'aide – `man`

Les pages du manuel sont organisées en section. Cette organisation peut varier d'un Unix à l'autre.

Par exemple, sous Linux, l'organisation est la suivante :

- section 1 : programmes et commandes du *shell* ;
- sections 2 et 3 : primitives du système et des bibliothèques ;
- section 4 : fichiers spéciaux ;
- section 5 : format des fichiers de configuration et conventions ;
- section 6 : jeux ;
- section 8 : commandes liées à l'administration du système.

Dans la documentation papier, on fait référence aux pages de manuel sous cette forme : « ls(1) ». Pour consulter la page du manuel d'une section particulière, il faut utiliser cette syntaxe :

```
$ man 5 passwd
```

La commande `apropos` affiche toutes les pages de manuel qui correspondent à un terme particulier.

Les *shells*

Plusieurs interpréteurs de commandes sont disponibles sous Unix.
Les *shells* les plus utilisés sont :

- le C Shell ;
- le Shell Korn ;
- le Shell Bourne.

Sous Linux, on utilise la plupart du temps le Bash (*Bourne Again Shell*).

Dans le monde Unix, les interpréteurs de commande sont fonctionnellement riches, plus encore sur les systèmes Linux où des fonctionnalités nouvelles sont disponibles (édition de la ligne de commande, historique, coloration, complétion, etc.).

Chaque *shell* dispose d'une syntaxe qui permet d'écrire des scripts évolués.

Variables d'environnement

Le *shell* permet de manipuler les variables d'environnement :

```
$ TEST="coucou !"
$ echo $TEST
coucou !
```

Par défaut, une variable est définie localement, c'est-à-dire qu'elle n'est pas transmise aux programmes appelés depuis le *shell*. Si elle doit l'être, il faut l'exporter ainsi :

```
$ export TEST
```

Les variables d'environnement sont très utilisées, voici quelques exemples :

- `$HOME` le répertoire de l'utilisateur ;
- `$SHELL` le *shell* utilisé ;
- `$USER` le nom de l'utilisateur connecté ;
- `$UID` l'UID de l'utilisateur.

La commande `set` permet de lister toutes les variables définies.

La variable `$PATH`

La variable `$PATH` contient, comme sous Windows, le chemin de recherche des commandes. Cependant, les répertoires sont séparés par des caractères « : ».

De plus, par défaut, le *shell* ne recherche par les programmes dans le répertoire courant (« . »), il faut pour cela l'ajouter explicitement au contenu de la variable `$PATH` ou appeler les commandes ainsi :

```
$ ./commande
```

Attention, la casse des caractères du nom des variables est importante.

Par exemple :

```
$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/form/bin
$ export PATH=.:$PATH
$ echo $PATH
.:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/form/bin
```

Développement des noms de fichier

Les caractères « `*` » et « `?` » permettent de spécifier des ensembles de fichiers tout comme sous DOS.
Cependant, contrairement à DOS, ce développement est réalisé par le *shell*.

Pour protéger un caractère spécial dans le nom d'un fichier (tel que `*`, `?`, `;`, `&`, `\`, `/`, etc.), on peut :

- le préfixer par le symbole « `\` » ;
- ou enclore le nom de fichier entre guillemets simples (`'`) ou doubles (`"`).

Par exemple :

```
$ mv toto.txt Un\ nom\ long\ \&\ bizarre.txt
$ mv toto.txt "Un nom long & bizarre.txt"
```

Redirections

La redirection des entrées et sorties des commandes est très utilisée sons Unix.

- `<` redirige l'entrée depuis le fichier indiqué ;
- `>` redirige la sortie vers le fichier indiqué ;
- `>>` redirection de la sortie avec ajout à la fin du fichier indiqué ;
- `|` (appelé tube ou *pipe*) redirige la sortie d'un programme vers l'entrée d'un autre.

Par exemple :

```
$ ls /etc > /tmp/liste-etc.txt
$ cat fich01.txt fich02.txt fich03.txt > tout.txt
```

Les commandes `more` et `grep` sont souvent utilisées ainsi :

```
$ ls -l /etc | more
$ set | grep HIST
```

`grep` recherche une chaîne de caractère dans un flux de données.
L'option `-i` permet de ne pas tenir compte de la casse des caractères

Redirection d'erreur

Il est possible de rediriger également la sortie d'erreur standard (qui est associée au *handle* de fichier 2) avec ces opérateurs : `2>` et `2>>`.

Par exemple :

```
$ ls
fichier
$ ls -l fichier toto
ls: toto: Aucun fichier ou répertoire de ce type
-rw-rw-r--    1 form     form         0 nov 19 19:43 fichier
$ ls -l fichier toto 2> /dev/null
-rw-rw-r--    1 form     form         0 nov 19 19:43 fichier
$ ls -l fichier toto > /dev/null
ls: toto: Aucun fichier ou répertoire de ce type
```

Le fichier spécial `/dev/null` est associé à un périphérique virtuel qui se comporte comme un puit sans fond.

La syntaxe « `2>&1` » permet d'associer la sortie d'erreur à la sortie standard.

Manipuler les fichiers

Les commandes `cp`, `mv` et `rm` permettent respectivement de copier, déplacer et supprimer un ou plusieurs fichiers.

Une option commune à ces commandes est `-i` (comme « interactif ») qui demande confirmation des actions supprimant (`rm`) ou écrasant (`cp` et `mv`) des fichiers. L'option opposée est `-f` (pour « force »).

Une autre option pour les commandes `cp` et `rm` est `-r` (pour « récursif ») qui permet d'agir sur un répertoire et tout son contenu.

Les programmes `mkdir` et `rmdir` sont utilisés pour créer ou supprimer des répertoires. `rmdir` ne supprime que des répertoires vides.

La ligne de commande suivante est souvent utilisée pour supprimer toute une arborescence de fichiers et répertoires :

```
$ rm -r -f répertoire
```

Ou, les options simples pouvant être agrégées :

```
$ rm -rf répertoire
```

L'éditeur *vi*

L'éditeur *vi* est le premier éditeur « pleine page » de l'histoire de l'informatique.

Il dispose des atouts suivants :

- éditeur pleine page (contrairement à *ed*) ;
- supporte un grand nombre de terminaux ;
- disponible sur tous les systèmes Unix ;
- peu gourmand en ressources ;
- il est puissant et productif.

Et voici ses inconvénients :

- d'un premier abord complexe ;
- les commandes peuvent paraître barbares.

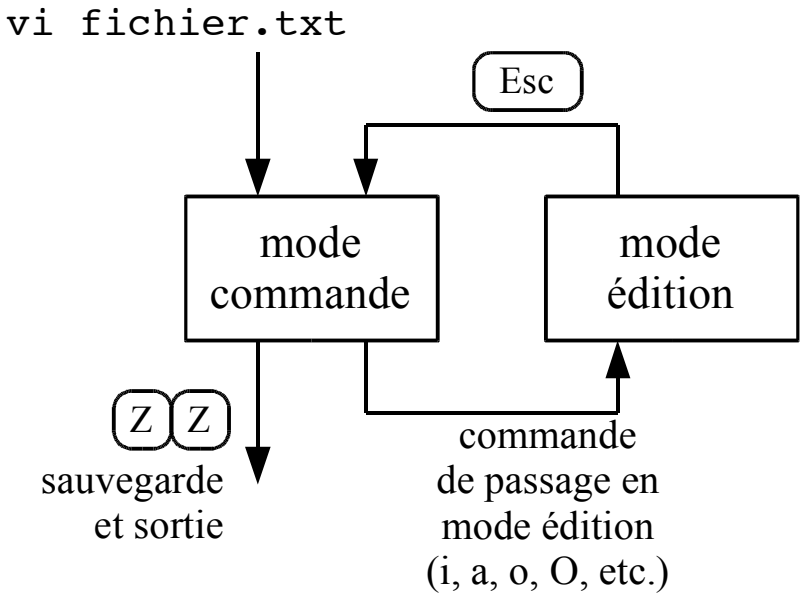
L'éditeur *vi*

Lorsque l'éditeur *vi* est démarré avec la commande `vi`, le mode actif est le mode de commande.

Il permet de se déplacer dans le texte, de réaliser des copier/coller, des recherches, des remplacements, etc.

Certaines commandes permettent de passer en mode édition dans lequel il est possible d'entrer du texte.

Pour quitter le mode édition, il faut utiliser la touche « Esc ».



L'éditeur *vi* – quelques commandes

Se déplacer	aller à la fin du fichier : « G » aller à la ligne N°*n* : « :*n* » puis « Entrée » rechercher une chaîne de caractères : « /*caractères* » puis « Entrée » occurrence suivante : « n » aller en début ou fin de ligne : « ^ » et « $ »
Supprimer	un caractère ou un mot : « x » ou « dw » une ligne : « dd » (puis, éventuellement, « p » pour la coller ailleurs) *n* caractères, mots ou lignes : « *n*x », « *n*dw » ou « *n*dd »
Éditer	insérer au point courant : « i » ajouter du texte en fin de ligne : « A » insérer une ligne avant ou après la ligne courante : « o » ou « O » remplacer un mot : « cw » écrire par-dessus du texte : « R »
Quitter	en sauvegardant : « ZZ » ou « :wq » puis « Entrée » sans enregistrer : « :q! » puis « Entrée »

Droits d'accès aux fichiers

À chaque fichier et répertoire du système est associé :

- un propriétaire (UID) ;
- un groupe propriétaire (GID) ;
- un ensemble de droits d'accès codés sur 2 octets.

Les droits d'accès à un fichier sont représentés de la manière suivante :

```
-  rwx  rwx  rwx
```

`type  propriétaire  groupe  autres`

Pour un répertoire, le droit `x` est celui qui permet de traverser le répertoire.

Le type du fichier n'est pas partie prenante du mécanisme des droits d'accès mais il est présenté par `ls` dans le même bloc d'informations.

Droits d'accès aux fichiers

Chaque type d'accès possède une valeur :

- `r` (lecture) a pour valeur 4
- `w` (écriture) a pour valeur 2
- `x` (exécution) a pour valeur 1

Types des fichiers :

- `-` fichier standard
- `d` répertoire
- `l` lien symbolique
- `c` fichier spécial caractère
- `b` fichier spécial bloc
- `f` FIFO (tube nommé)
- `s` *socket* de communication

Droits d'accès aux fichiers – `chmod`

La commande `chmod` permet de modifier les droits d'accès aux fichiers et répertoires. Ils peuvent être spécifiés en octal (un chiffre par groupe de droits) ou avec la syntaxe suivante :

`[augo](+|-|=)[rwxs]`

- `a` tous (*all*)
- `u` propriétaire (*user*)
- `g` groupe propriétaire
- `o` autres (*others*)

L'option `-R` (récursif) permet d'appliquer la modification à un répertoire et tout son contenu.

Exemples :

```
$ chmod 664 fichier1 fichier2
$ chmod 755 répertoire
$ chmod a+x commande
$ chmod -R go-rwx répertoire
```

Droits d'accès aux fichiers – `chown` et `chgrp`

Les commandes `chown` et `chgrp` permettent de modifier le propriétaire et le groupe propriétaire de fichiers et répertoires.

L'utilisation de ces commandes est relativement restreinte pour les utilisateurs qui ne sont pas *root*, ils ne peuvent changer que le groupe propriétaire d'un fichier (et ils doivent être membre de ce groupe).

L'option `-R` (récursif) permet d'appliquer la modification à un répertoire et tout son contenu.

Exemples :

```
# chown snameche fichier1
# chown snameche:staff fichier2
# chgrp -R users répertoire
```

Les processus

Un processus définit tout l'environnement nécessaire au fonctionnement d'un programme. Cela comprend :

- le code du programme ;
- l'espace mémoire pour les données et la pile d'exécution ;
- les variables d'environnement ;
- le numéro de processus (PID) ;
- le numéro de processus père (PPID) ;
- les identifiants d'utilisateur et de groupe propriétaire (UID et GID) ;
- les descripteurs de fichiers.

Chaque processus possède un père qui l'a été créé, excepté le processus dont le numéro est 1. Celui-ci exécute la commande `init` qui sert à démarrer le système Unix et à charger tous les programmes nécessaire pour assurer son fonctionnement.

Les commandes `ps` et `kill` servent à manipuler les processus.

Les processus – `ps` et `top`

La commande `ps` affiche la liste des processus en mémoire.

Voici quelques options :

- `-a` affiche aussi les processus des autres utilisateurs
- `-x` affiche aussi les processus qui n'ont pas de terminal de contrôle.
- `-u` indiquer le propriétaire de chaque processus
- `-f` affiche l'arbre généalogique des processus

Exemples :

```
$ ps -fax
$ ps fax
```

La commande `top` affiche un écran actualisé régulièrement qui présente les processus et statistiques générales du système (*uptime*, mémoire utilisée, charge, etc.).

Les processus – `kill`

La commande `kill` est utilisée pour envoyer un signal à un processus.

Un signal est identifié par sa valeur numérique ou son identifiant. Les processus auxquels est destiné le signal sont identifiés par leur PID.

L'option `-l` permet de lister tous les signaux disponibles. Parmi ceux-ci, on retiendra :

- `HUP` (1), relire une nouvelle configuration et réouvrir les fichiers ;
- `INT` (2), signal envoyé par la combinaison de touches « Ctrl-c » ;
- `KILL` (9), arrête « violemment » un processus, signal traité par le système ;
- `TERM` (15), envoyé par défaut, demande de terminaison ;
- `SEGV` (11), erreur de segmentation (caractéristique d'un *bug* logiciel) ;
- `BUS` (7), erreur de bus (caractéristique d'un problème matériel) ;
- `USR1` (10) et `USR2` (12), à la discrétion du programmeur.

Exemples :

```
$ kill 123
$ kill -9 123
```

Programmes en tâche de fond

Le caractère « `&` » utilisé à la fin d'une ligne précise au *shell* que la commande doit être exécutée en tâche de fond. Le programme est lancé et son numéro de processus (PID) est renvoyé.

Par exemple :

```
$ find / > listing 2> /dev/null &
[1] 15596
$
```

Lorsque la commande est terminée, un message en informe l'utilisateur :

```
[1]+  Exit 1                  find / >listing 2>/dev/null
```

Cependant, lorsque la session est fermée, le signal `HUP` est envoyé à tous les processus actifs attachés au terminal, ce qui provoque généralement leur terminaison prématurée.

Pour éviter cela, on utilise la commande `nohup` ainsi :

```
$ nohup find / > listing 2> /dev/null &
```

Transférer des fichiers

Pour échanger des fichiers, les procédés suivants peuvent être utilisés :

- les mtools, lorsqu'ils sont disponibles, permettent de travailler efficacement avec des disquettes ;
- les CD sont gérés via les commandes `mount`, `umount` et `eject` ou par des mécanismes de type *automounter* ;
- les systèmes de fichiers réseau (NFS, SMB, etc.) rendent l'accès transparent aux ressources partagées mais leur configuration doit être réalisée par l'administrateur du système ;
- l'échange de fichiers par le réseau peut aussi se faire par FTP ou avec les commandes `scp` (*Secured Copy*) et `smbclient` (du projet Samba).

Exemples :

```
$ mcopy a:application.tar.gz /tmp
$ mount /mnt/cdrom; copy /mnt/cdrom/readme.txt .; eject cdrom
$ scp toto.txt snameche@machine.reseau.fr:/chemin/complet
$ scp -r machine:chemin/relatif .
```

Lire et envoyer des émails

La commande `mail` fournit une interface basique pour lire et écrire des émails. Pour lire les messages, il suffit d'utiliser cette commande sans argument :

```
$ mail
Mail version 8.1 6/6/93.  Type ? for help.
"/var/spool/mail/form": 2 messages 1 new 2 unread
 U  1 form@jo.luke.net   Mon Nov 29 10:40  19/581  "TEST"
>N  2 form@jo.luke.net   Mon Nov 29 10:45  32/656  "Output from your job"
& 1
Message 1:
From form@jo.luke.net  Mon Nov 29 10:40:32 2004
Date: Mon, 29 Nov 2004 10:40:32 +0100
From: Formation <form@jo.luke.net>
To: form@jo.luke.net
Subject: TEST

Bla bla.

& d
& q
Saved 1 message in mbox
```

Lire et envoyer des émails

Un message peut être envoyé de manière interactive ou non :

```
$ mail sebastien@toto.fr
Subject: Test II
Bla.

-- S. Nameche
.
$ cat /var/log/backup.log | mail -s "Log sauv." admin@toto.fr
```

D'autres logiciels de messagerie (plus évolués) sont souvent disponibles.
Par exemple, sous Linux :

- en console : Mutt, pine, elm, etc. ;
- sous XWindow : evolution, Sylpheed, Mozilla, etc.

Il est possible de rediriger les émails d'un utilisateur Unix en créant un fichier nommé `.forward` dans son répertoire. Ce fichier contient l'adresse émail vers laquelle les messages seront renvoyés.

Le planificateur de tâches – `cron`

Le démon *crond* a pour fonction d'exécuter régulièrement des commandes à des heures programmées. Ce démon s'appuie sur le fichier `crontab` pour déterminer quelles sont les commandes à exécuter et à quelle heure.

Chaque utilisateur dispose d'un fichier `crontab` personnel. Il contient une ligne par commande programmée. Le format de chaque ligne est le suivant :

```
mm   hh   dM   MM   dw   commande
```

- `mm` minutes (0-59)
- `hh` heure (0-23)
- `dM` jour dans le mois (1-31)
- `MM` mois (1-12)
- `dw` jour de la semaine (0-7, 0 et 7 représentent dimanche)

Le planificateur de tâches – `crontab`

La commande `crontab` est utilisée pour manipuler le fichier crontab.

L'option `-e` permet de modifier le fichier `crontab`, l'option `-l` de l'afficher et l'option `-r` de le supprimer.

Voici un exemple de fichier `crontab` :

```
# mm hh dM MM  dw  commande
  30  0  1  *   *  cat /var/log/appli.log | mail admin
  42  5  *  * 1-5  backup.sh > /var/log/backup.log
```

Autres commandes

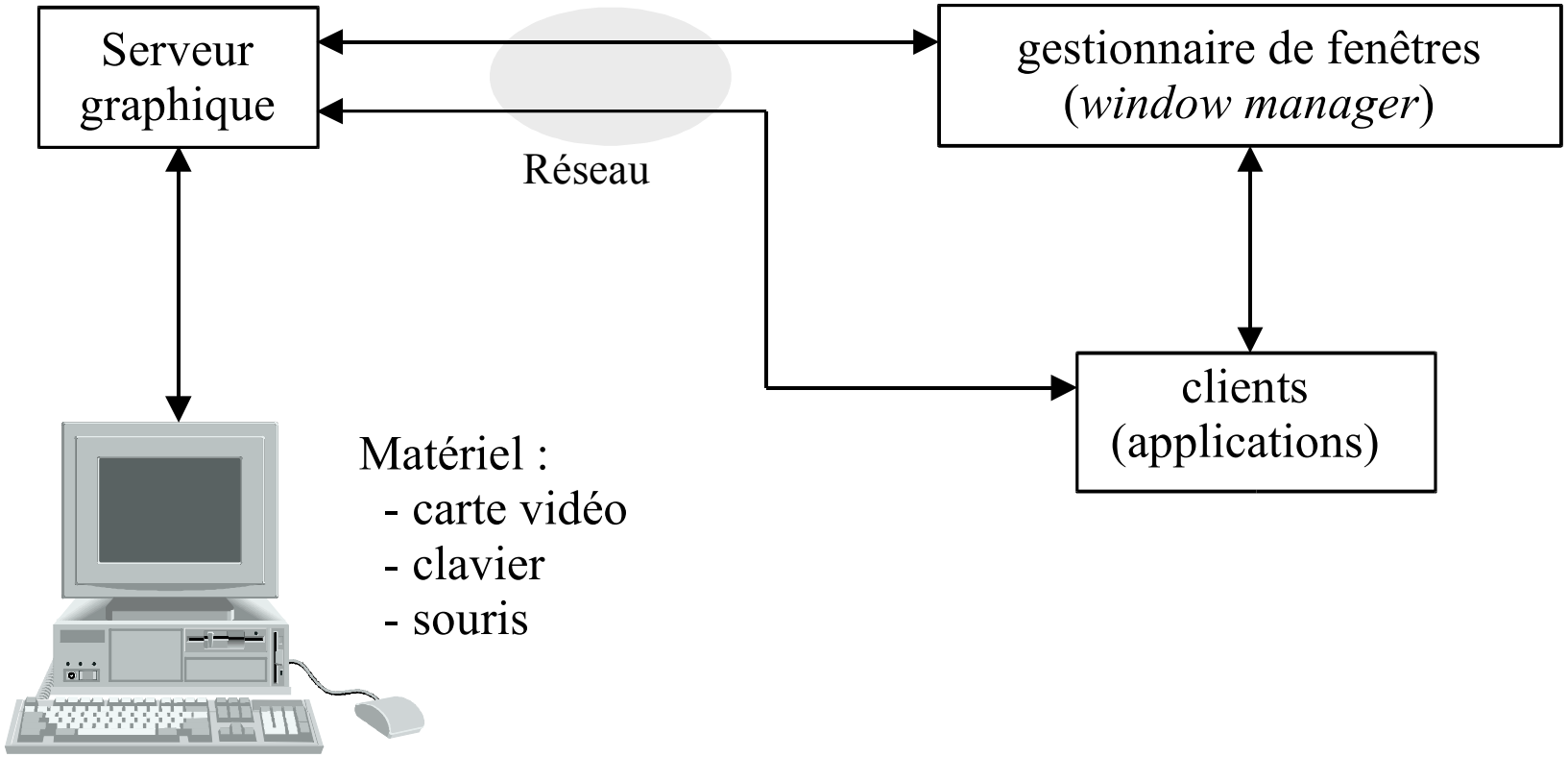
Voici quelques commandes supplémentaires souvent utilisées :

Créer des liens symboliques	`ln -s`
Changer de mot de passe	`passwd`
Devenir super-utilisateur	`su -`
Obtenir la date et l'heure	`date`
Examiner l'espace disque	`df` et `du`
Déterminer le type de fichiers	`file`
Créer des archives	`tar` et `cpio`
Compresser des fichiers	`compress` (`.Z`), `gzip` (`.gz`), `bzip2` (`.bz2`)
Décompresser des fichiers	`uncompress`, `gunzip` et `bunzip2`
Redirection console et fichier	`tee`
Formater une disquette	`mformat a:` ou `mkfs -t vfat /dev/fd0`

XWindow

X-Window (ou « X ») est une sur-couche graphique orientée réseau.

Un environnement de travail sous X est organisé ainsi :



Exécuter une application X distante

Afin d'afficher en local une application qui s'exécute à distance, deux étapes sont nécessaires :

1) Autoriser les connexions au serveur X en utilisant la commande `xhost`. Par exemple, pour autoriser toutes les connexions :

```
local$ xhost +
```

2) Indiquer au programme distant quel serveur X contacter pour l'affichage en utilisant la variable d'environnement `$DISPLAY` ou l'option `-display` reconnue par toutes les applications X :

```
distant$ DISPLAY=machine_locale:0
distant$ export DISPLAY
distant$ xterm &
```

ou :

```
distant$ xterm -display mamachine:0 &
```

Exécuter une application X distante

Voici un exemple qui permet d'exécuter un programme graphique en tant qu'utilisateur *root* :

```
$ xhost +
access control disabled, clients can connect from any host
$ su -
Password:
# DISPLAY=:0
# export DISPLAY
# xeyes &
```

SSH permet de palier à de nombreux problèmes de sécurité liés à l'exécution d'applications X distantes.

```
snameche@station$ ssh -X -C oracle@serveur.domaine.fr
oracle@serveur# cd /mnt/cdrom; ./runInstaller &
```

L'option `-X` active l'encapsulation du protocole X et `-C` la compression du flux SSH. L'utilisation de la commande `xhost` n'est pas requise dans ce cas.

Références

Livres

Aux Éditions O'Reilly :

« Introduction à Unix », 5ème édition
de Jerry Peek, Grace Todino et John Strang
(l'aide-mémoire joint à ce support de cours provient de cet ouvrage)

« Linux en concentré », 4ème édition
de Ellen Siever, Stephen Figgins et Aaron Weber

« Le système Linux », 4ème édition
de Matt Welsh, Matthias Kalle Dalheimer, Terry Dawson et Lar Kaufman

« L'éditeur vi, précis & concis »
de Arnold Robbins

Chez Dunod

« UNIX: Programmation et communication »
de Jean-Marie Rifflet et Jean-Baptiste Yunes