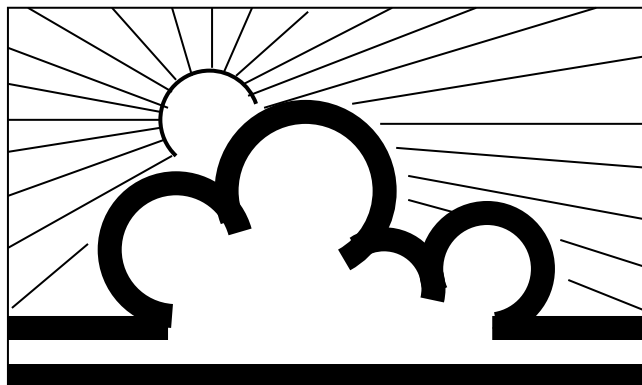

Initiation UNIX

Licence d'informatique 2001 – 2002

Thierry Besançon



Université de Versailles – Saint-Quentin-en-Yvelines

Les animateurs de ce cours peuvent être joints aux adresses suivantes :

Thierry.Besancon@prism.uvsq.fr

Ivan.Djelic@prism.uvsq.fr

Alexis.Troubnikoff@prism.uvsq.fr

Ce cours est disponible au format PDF sur le web à l'URL :

<http://www.prism.uvsq.fr/~thb/licence-unix-2001-2002.pdf>

La validité de cet URL n'est pas garantie au delà du 31 décembre 2001.

"...the number of UNIX installations has grown to 10, with more expected..."

- Dennis Ritchie and Ken Thompson, June 1972

Table des matières

N° de transparent

Chapitre 1	Avant propos – Informatique à l'UVSQ	1
§ 1	La charte informatique	1
§ 2	Qui gère les ressources informatiques pédagogiques ?	2
§ 3	Que signaler ? – Comment le signaler ?	3
Chapitre 2	UNIX : Généralités – Historique	4
§ 1	UNIX, un système d'exploitation	4
§ 2	Les UNIX du marché	8
Chapitre 3	Premiers contacts	10
§ 1	Votre compte : login, mot de passe	11
§ 2	Principales règles sur les mots de passe	12
§ 3	Changer son mot de passe	13
§ 4	Connexion sur les terminaux	14
§ 5	L'interface graphique	18
§ 6	Les langages de commande : les shells	22
§ 7	Formes générales des commandes Unix	28
§ 8	Documentation Unix en ligne : man	30
Chapitre 4	Manipulations de base sur les fichiers Unix	35
§ 1	Arborescence de fichiers	35
§ 2	Rappel sur les options de commande Unix	42
§ 3	Liste des fichiers : ls	43
§ 4	Contenu d'un fichier texte : cat, more	49
§ 5	Destruction d'un fichier : rm	51
§ 6	Duplication d'un fichier : cp	53
§ 7	Déplacement et renommage d'un fichier : mv	55
§ 8	Création de répertoires : mkdir	57
§ 9	Suppression de répertoires : rmdir	58
§ 10	Position dans les répertoires : cd, pwd	59
§ 11	Editeur de fichier texte – vi, emacs	60
Chapitre 5	Commandes auxiliaires de manipulation de fichiers Unix	63
§ 1	Droits d'accès : chmod	63
§ 2	Recherche de chaînes de caractères : grep	69
§ 3	Modification à la volée de contenu de fichiers : sed	72
§ 4	Tri d'un fichier : sort	73
§ 5	Comptage de caractères, mots, etc. : wc	74
§ 6	Comparaison de fichiers : diff	75
§ 7	Manipulation des noms de fichiers : basename, dirname	76
§ 8	Manipulation sur des lignes de fichiers : head, tail	78
§ 9	Recherche de fichiers : find	81
§ 10	Compression de fichiers : (un)compress, g(un)zip	84
§ 11	Archivage : tar	85
§ 12	Liens sur fichiers : ln, ln -s	87
§ 13	Langage plus complet : awk	91
§ 14	Transfert de fichiers depuis et vers un lecteur de disquettes : mcopy	94
Chapitre 6	Pratique du Bourne shell	95
§ 1	Principe d'exécution d'une commande	95
§ 2	Caractères spéciaux du shell : métacaractères	96
§ 3	Contrôle des commandes lancées : &, fg, bg, kill, ^C, ^Z	97
§ 4	Contrôle des processus : ps, kill, top	103
§ 5	Quote characters : ', ", \	111
§ 6	Caractères de redirection : <, >, >>, <<, \, , 2>, >&	113
§ 7	Désignation des fichiers par leurs noms : *, ?, [], [^]	126
§ 8	Variables	128

§ 9	Variables d'environnement	131
§ 10	Ordre d'évaluation de la ligne de commande	135
§ 11	Double évaluation	138
§ 12	Quitter un shell : <code>exit</code> , <code>Ctrl-D</code>	140
Chapitre 7	Programmation en Bourne shell	141
§ 1	Caractéristiques d'un shell script	142
§ 2	Structure d'un shell script	143
§ 3	Passage de paramètres à un shell script : <code>\$1</code> à <code>\$9</code>	149
§ 4	Liste des paramètres d'un shell script : <code>\$*</code> , <code>\$@</code>	155
§ 5	Variables prédéfinies : <code>\$?</code> , <code>\$!</code> , <code>\$\$</code>	156
§ 6	Commandes internes du shell : <code>builtins</code> , <code>type</code>	157
§ 7	Commande d'affichage : <code>echo</code>	158
§ 8	Entrée interactive : <code>read</code>	161
§ 9	Structure <code>if - then - else</code>	164
§ 10	Structure <code>case</code>	168
§ 11	Commande <code>test</code>	170
§ 12	Structure de boucles : <code>while</code> , <code>for</code> , <code>until</code>	175
§ 13	Contrôle du flux d'exécution : <code>break</code> , <code>continue</code>	179
§ 14	Redirection dans les boucles	181
§ 15	Fonctions shell	185
§ 16	Code de retour d'un shell script : <code>exit</code>	188
§ 17	Traitement des signaux : <code>trap</code>	189
§ 18	Debugging d'un shell script : <code>set -x</code>	193
Chapitre 8	Travail en réseau à l'UVSQ et sur Internet	195
§ 1	Raccordement Internet de l'UVSQ (plan de 1999)	195
§ 2	Accès à vos fichiers, NFS, <code>df</code> , <code>du</code> , <code>quota</code>	197
§ 3	Impression : <code>lpr</code> , <code>lpq</code> , <code>lprm</code>	204
§ 4	Nom de machine : <code>uname</code> , <code>hostname</code>	205
§ 5	Tests de connectivité : <code>traceroute</code> , <code>ping</code>	206
§ 6	Transfert de fichiers entre machines, <code>ftp</code>	208
§ 7	Connexion shell sur des machines distantes : <code>telnet</code> , <code>ssh</code>	211
§ 8	Liste d'utilisateurs connectés : <code>users</code> , <code>who</code> , <code>w</code>	213
§ 9	Courrier électronique, adresse, <code>mutt</code> , <code>netscape</code>	214
§ 10	Web, URL, <code>lynx</code> , <code>netscape</code>	215
Chapitre 9	Système de multifenêtrage : X	219
§ 1	Caractéristiques de X	221
§ 2	Clavier	223
§ 3	Souris	224
§ 4	Ecran	225
§ 5	Fenêtre	226
§ 6	Icône	227
§ 7	DISPLAY	228
§ 8	Architecture de X	229
§ 9	Serveur X	230
§ 10	Clients X	231
§ 11	Gestionnaire de fenêtres	233
§ 12	Spécification des options aux clients X	237
§ 13	Spécification des couleurs aux clients X	239
§ 14	Spécification des polices de caractères	241
§ 15	Personnalisation – Ressources X	243
§ 16	Autorisation d'accès	250
§ 17	Utilisation répartie de X	258
§ 18	XDM : X Display Management	262

Chapitre 1 : Avant propos – Informatique à l'UVSQ

§ 1.1 La charte informatique

La charte informatique (annexe A) définit le règlement intérieur à l'Université en ce qui concerne l'Informatique.

Vous devez vous y conformer inconditionnellement.

Cette année, l'Université prendra des sanctions administratives envers tout étudiant violant de façon mineure la charte.

Cette année, l'Université déposera systématiquement plainte envers tout étudiant contrevenant et impliqué dans un «piratage».

§ 1.2 Qui gère les ressources informatiques pédagogiques ?

Le Centre de Services Informatiques (dit **CSI**) gère les machines des étudiants.

Vos interlocuteurs :

◇ Thierry Caillet

bureau : pièce 200, 2^e étage, bâtiment Descartes

email : Thierry.Caillet@csi.uvsq.fr

tél : 01 39 25 78 86

◇ Rémy Card

bureau : pièce 202, 2^e étage, bâtiment Descartes

email : Remy.Card@csi.uvsq.fr

tél : 01 39 25 78 86

§ 1.3 Que signaler ? – Comment le signaler ?

Vous **devez** signaler :

- tout problème de compte ; dans ce cas, se présenter aux bureaux 200 ou 202 **muni de sa carte d'étudiant** ; rien ne sera fait si vous ne justifiez pas de votre inscription à l'Université par votre carte d'étudiant...

Faire au plus vite si vous soupçonnez que votre compte est piraté.

- problème sur les imprimantes : cartouche d'encre vide, bourrage papier, etc. ; dans ce cas un email à `sysadm@ens.uvsq.fr`
- problème sur le poste de travail : terminal en mode inhabituel, clavier cassé, souris hors service, etc. ; dans ce cas un email à `sysadm@ens.uvsq.fr`
- problème anormal avec un logiciel : le logiciel ne fonctionne plus comme d'habitude, un logiciel a disparu, le logiciel ne fonctionne pas du tout comme le précise la documentation, etc. ; dans ce cas un email à `sysadm@ens.uvsq.fr`

Chapitre 2 : UNIX : Généralités – Historique**§ 2.1 UNIX, un système d'exploitation**

Les missions d'un système d'exploitation sont :

- mise à disposition de ressources matérielles : espace disque, temps d'exécution sur le microprocesseur central, espace mémoire, etc.
- partage équitable de ces ressources entre les utilisateurs pour atteindre le but de système multi-utilisateurs

◇ Terminologie :

Mono utilisateur	Une seule personne utilise l'ordinateur
Multi utilisateur	Plusieurs personnes peuvent utiliser le système en même temps. Le système s'assure qu'un utilisateur n'interfère pas sur un autre.
Mono tâche	Un seul processus tourne à un instant.
Multi tâche	Plusieurs processus donnent l'impression de tourner en même temps.
Multi tâche préemptif	L'OS détermine quand un processus a eu assez de temps CPU.
Multi tâche non pré-emptif	Le processus détermine lui même quand il a eu assez de temps CPU.

◇ Exemples :

MS DOS	mono utilisateur, mono tâche
Windows 95/98	mono utilisateur, multi tâche non préemptif
Windows NT	mono utilisateur, multi tâche préemptif
OS/2	mono utilisateur, multi tâche préemptif
UNIX	multi utilisateur, multi tâche préemptif

◇ Concepts novateurs d'UNIX :

« UNIX est construit autour d'une idée forte : la puissance d'un système provient plus des relations entre les programmes que des programmes eux-mêmes. Beaucoup de programmes UNIX font, de façon isolée des traitements triviaux ; combinés avec d'autres, ils deviennent des outils généraux et performants.

La solution d'un problème sous UNIX ne passe pas forcément par l'écriture d'un programme spécifique mais souvent par une utilisation combinée et élégante des outils standard. »

§ 2.2 Les UNIX du marché

Quelques Unix de constructeurs de matériels :

CRAY (Unicos ?. ?), **DEC** (Digital Unix 4.0, 5.0), **HP** (HP-UX 9.07, HP-UX 10.20, HP-UX 11.x), **IBM** (AIX 3.2.5, AIX 4.x.y, 5.x), **SGI** (IRIX 6.x.y), **SUN** (SunOS 4.1.4, Solaris 2.x.y puis Solaris 7 et 8)

Les autres Unix ne sont pas conçus par des constructeurs de matériels :

Santa Cruz (SCO 5.5), **Novell** (Unixware 4.0), **Linux** (noyau 2.x.y, nombreuses distributions), **FreeBSD** (FreeBSD 4.x.y), **NetBSD** (NetBSD 1.x.y), **OpenBSD** (OpenBSD 1.x.y)

Cf l'annexe pour un arbre généalogique d'UNIX.

Du point de vue de l'utilisateur, les divers UNIX se ressemblent beaucoup.

◇ Tentatives d'unification

- *System V Interface Definition* de AT&T (SVID, SVID2, SVID3 en 1989)
- IEEE POSIX (POSIX1003.1 en 1990)
- X/OPEN Portability Guide (XPG4 en 1993) du consortium X/OPEN (créé en 1984)

◇ La concurrence

Microsoft Windows NT 4.0 ou 2000 ou XP.

Chapitre 3 : Premiers contacts

Avant de commencer : n'ayez pas peur d'expérimenter. Le système ne vous fera pas de mal. Vous ne pouvez rien abîmer en utilisant le système. UNIX, par conception, possède des notions de sécurité, afin d'éviter aux utilisateur «normaux» de le déconfigurer.

§ 3.1 Votre compte : *login*, mot de passe

Un utilisateur Unix est équivalent à :

- un identificateur (sur 8 lettres en général), son «nom» au sens informatique ; appelé **login** ;
- un mot de passe **confidentiel** ;

Gare aux sanctions en cas d'«amusement» avec le compte d'un autre utilisateur !

Cf la charte informatique en annexe.

En cas de problème avec votre compte, contactez les interlocuteurs du CSI (cf chapitre 1).

§ 3.2 Principales règles sur les mots de passe

- un mot de passe ne se prête pas !
- un mot de passe ne s'oublie pas !
- un mot de passe n'est pas facile à trouver ! :
 - évitez qu'il ne se rapporte pas à vous (nom, voiture, chien)
 - évitez les mots dans des dictionnaires
 - évitez les prénoms
 - il doit comporter au moins 6 caractères, en général 8
 - les majuscules et les minuscules sont différenciées
 - utiliser des chiffres et des caractères spéciaux : par exemple Kpiten[, &7oubon, etc.

Cf

<http://www.cru.fr/securite/Cours/mot-de-passe-jplg.html>

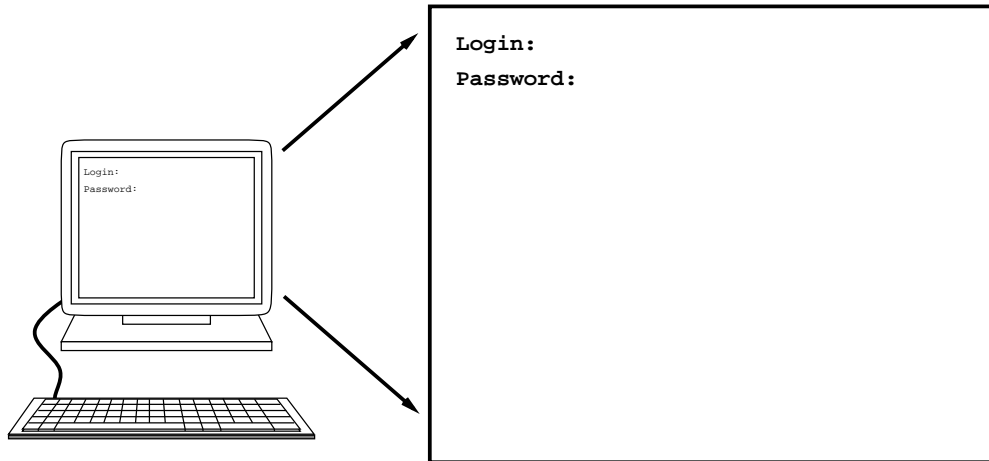
§ 3.3 Changer son mot de passe

La commande pour changer son mot de passe sur les machines de licence UVSQ est `passwd`.

Cf l'annexe sur les mots de passe

§ 3.4 Connexion sur les terminaux

◇ La connexion peut se faire sur un terminal texte.

**Attention :**

Pour une connexion sur une console, il ne faut pas entrer son nom de login alors que la touche **CAPS LOCK** est activée. En effet, si le nom de login est tout en majuscules, l'ordinateur croira que le terminal ne connaît pas les lettres minuscules et vous vous retrouverez dans une session où l'on ne distinguera pas minuscules des majuscules.

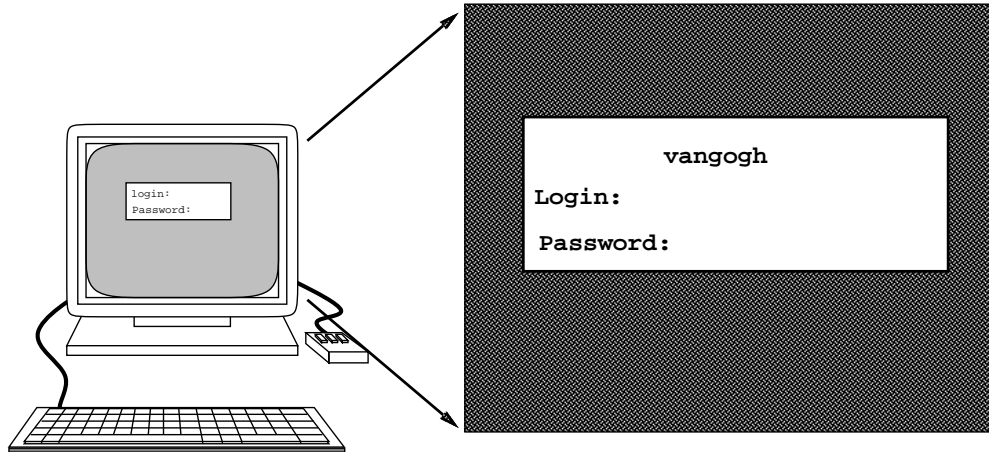
Remèdes :

- se déconnecter (la meilleure méthode)
- ou taper la commande `stty -lcase` ou `STTY -LCASE`

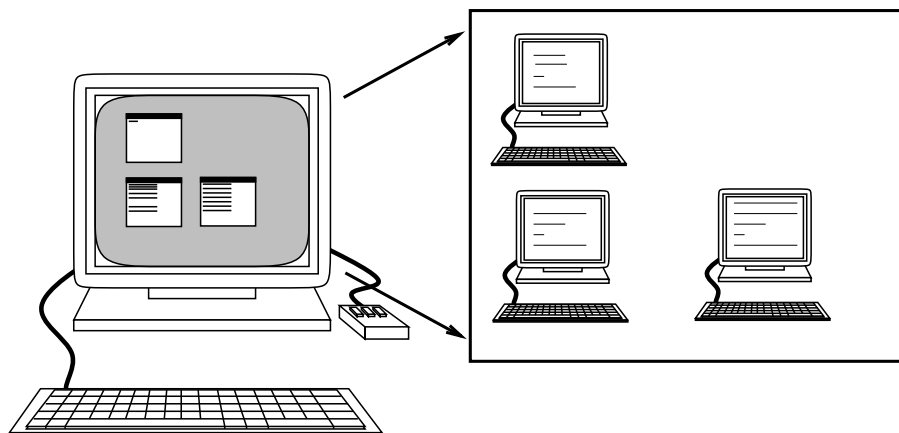
Attention bis :

UNIX fait la différence entre les lettres minuscules et majuscules au niveau des noms de commande et des noms de fichiers.

◇ La connexion peut se faire sur un terminal graphique.



Une fois connecté via l'interface graphique, on utilisera principalement un programme d'émulation de terminal de type texte qui fournit dans une fenêtre une connexion comme sur un terminal texte :



L'émulateur de terminal s'appelle `xterm`.

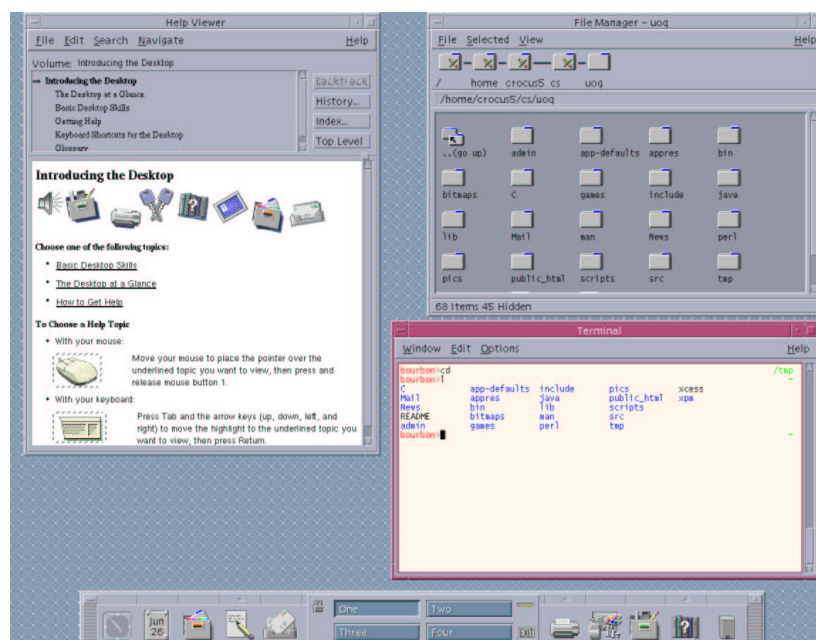
§ 3.5 L'interface graphique

Unix dispose d'un grand nombre d'interfaces graphiques comparé à des systèmes comme Windows ou Macintosh :

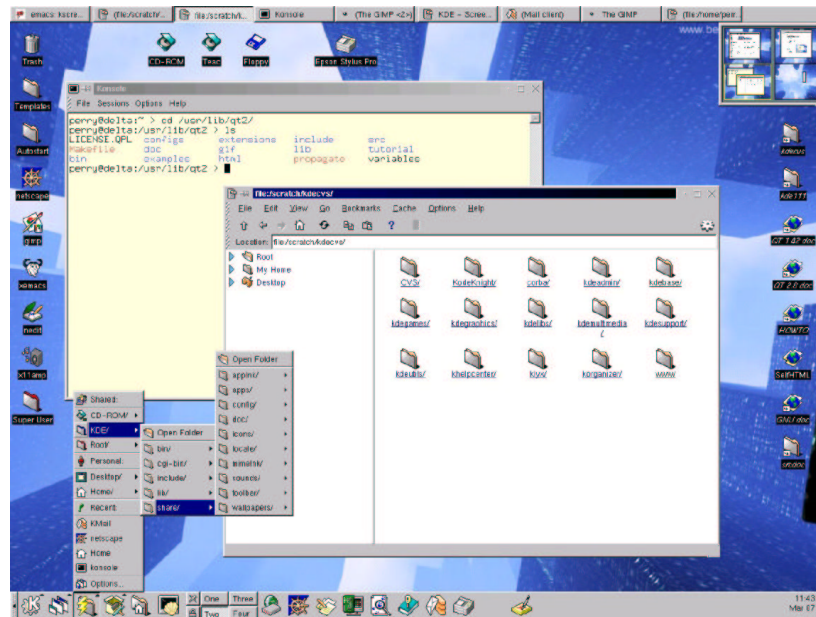
SunView, NeWS, OpenWindows, View, CDE, KDE, GNOME, Berlin, etc.

Cf le site <http://www.plig.org/xwinman/>

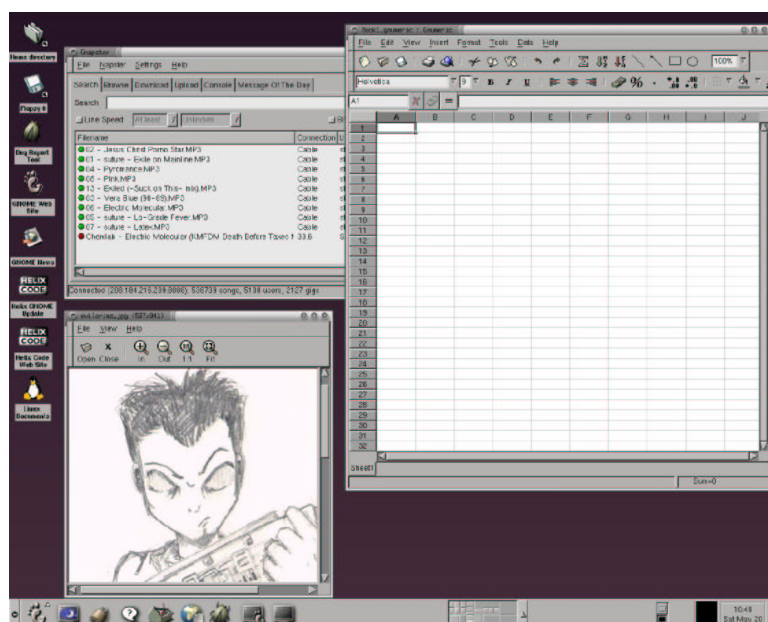
Bureau de travail sous CDE :



Bureau de travail sous KDE :



Bureau de travail sous GNOME :



§ 3.6 Les langages de commande : les shells

A l'origine, des teletypes puis des consoles texte.

⇒ l'interaction de base se fait au moyen de phrases à taper sur un clavier (par opposition aux interfaces graphiques à la Windows ou de Macintosh).



A gauche, une console texte DIGITAL VT100.

A droite, un teletype DIGITAL (genre de machine à écrire).

Le shell est un programme qui permet la saisie et l'interprétation de ce qui est tapé.
Le shell est juste une interface avec le système.

MS-DOS comporte un shell aux possibilités restreintes par rapport aux shells Unix.

Le shell est aussi un vrai langage de programmation, interprété (non compilé)
offrant les structures de base de programmation de tout autre langage.

Sous Unix, le shell est un programme au même titre qu'un autre. Le shell de travail est **interchangeable** par un autre shell (à la syntaxe près comme de bien entendu).

Les shells les plus répandus :

Shell	Program	Description
Bourne shell	sh	disponible sur toute plateforme Unix
C shell	csh	shell développé pour BSD
Korn shell	ksh	Bourne shell amélioré par AT&T
Bourne again shell	bash	Shell distribué avec linux ; version améliorée de sh et csh

Dans ce cours, on distinguera le **shell de programmation** (car on peut programmer grâce à un interpréteur de commandes s'il est bien pensé) du **shell de travail** lors d'une session interactive. Les 2 shells n'ont pas de raison d'être identiques (cf plus loin sur ce que cela implique).

Tous les shells se présentent sous la même forme à l'écran lorsqu'ils fonctionnent : une chaîne de caractères qui affiche que le shell attend que l'utilisateur tape quelque chose au clavier ; c'est le **prompt**.

Pour ce cours, on utilisera le caractère % pour désigner le prompt d'un utilisateur normal.

La suite du support de cours comportera des exemples comme :

```
% ls
```

Il ne faudra jamais taper la chaîne de prompt lorsque vous testerez par vous mêmes les commandes indiquées.

Pour terminer une session shell, on tape la commande commune à tous les shells :

```
% exit
```



Ken Thompson et Dennis M. Ritchie, les parents d'Unix

On notera les teletypes 33 ! Comme quoi pas besoin d'écran 20 pouces ou de carte video 3D pour travailler brillamment !

§ 3.7 Formes générales des commandes Unix

Une commande Unix \equiv un ensemble de mots séparés par des caractères blancs (caractère espace, tabulation)

Le premier mot : le nom de la commande

Le reste des mots : les paramètres de la commande

Particularités de certains mots : des options qui changent le comportement de la commande

En pratique on trouvera donc écrit :

commande [options] parametres

Les 2 crochets [et] indiquent que les options ne sont pas obligatoires. Il ne faut pas taper ces crochets sur la ligne de commande.

◇ Comment spécifie-t-on une option ?

Une option est quelque chose de prévu par le programme \Rightarrow c'est le programmeur qui aura toujours le dernier mot.

Il reste une tendance générale : Une option est introduite par le signe - et est souvent constituée d'une seule lettre comme par exemple -a. (mais attention aux exceptions nombreuses)

Souvent on pourra cumuler des options :

`ls -a -l \equiv ls -al`

Souvent (mais pas tout le temps), l'ordre des options n'a pas d'importance. (cf `getopt(1)` ou `getopt(3)`)

`ls -a -l \equiv ls -al \equiv ls -la \equiv ls -l -a`

§ 3.8 Documentation Unix en ligne : `man`

Il existe une documentation électronique accessible pendant le fonctionnement du système : c'est l'aide en ligne.

La commande donnant l'aide est **man**. Elle donne accès aux pages de manuel des commandes Unix qui sont réparties selon des sections comme suit :

- section 1 ≡ commandes normales
- section 2 ≡ appels systèmes
- section 3 ≡ fonctions de programmation C
- section 4 ≡ périphériques et pilotes de périphériques
- section 5 ≡ format de fichiers
- section 6 ≡ jeux
- section 7 ≡ divers
- section 8 ≡ commandes de gestion du système

Les numéros de section apparaissent dans de nombreux exemples.

Lorsque l'on verra `getopt (3)`, il faudra se reporter à la commande `getopt` de la section **3** du manuel.

Syntaxe de la commande `man` :

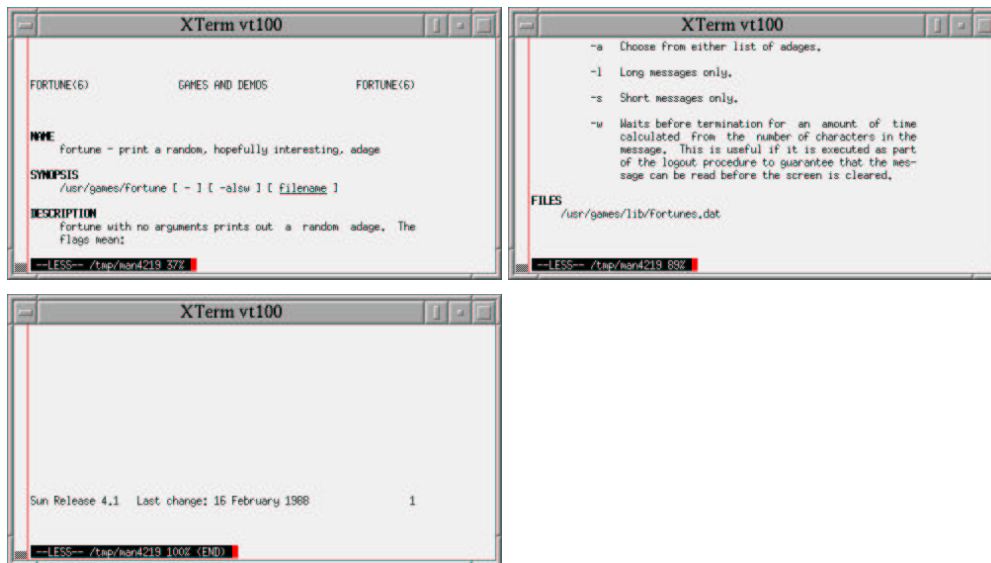
`man [numero de section] commande`

(les 2 crochets [et] indiquent que les options ne sont pas obligatoires ; il ne faut pas taper les crochets)

⇒ Inconvénient : il faut connaître le nom de la commande (nom anglais très souvent)

L'aide est plus là pour se rappeler les nombreuses options des commandes et leurs syntaxes particulières.

Exemple : `man fortune` renvoie :



Quelques pages de manuel intéressantes :

- la page de la commande `man` que l'obtient par `man man`
- les pages d'introduction de chaque section que l'on obtient par `man 1 intro`, `man 2 intro`, `man 3 intro`, etc.

```
% man 8 intro
```

```
INTRO(8)                      FreeBSD System Manager's Manual          INTRO(8)

NAME
    intro - introduction to system maintenance and operation commands

DESCRIPTION
    This section contains information related to system operation and
    ...
```

Quand on ne connaît pas le nom de la commande, on peut demander les noms des commandes dont le descriptif contient une certaine chaîne de caractères :

```
man -k chaîne
```

Exemple :

```
% man -k tune
xvidtune(1)      - video mode tuner for XFree86
xvidtune(1)      - video mode tuner for XFree86
fortune(6)       - print a random, hopefully interesting, adage
tunefs(8)        - tune up an existing file system
```

Chapitre 4 : Manipulations de base sur les fichiers Unix

§ 4.1 Arborescence de fichiers

Sur Unix, les fichiers sont identifiés par leur nom de fichier, qui peuvent contenir n'importe quel caractère (sauf /) et peuvent faire jusqu'à 256 caractères de long, voire plus selon les Unix.

Au concept de fichier est associé la notion de répertoire (en anglais directory). Un répertoire est simplement une collection de fichiers organisée de manière arborescente.

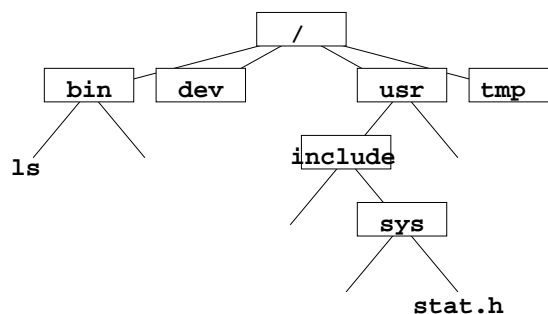
Un fichier peut être référencé par son chemin d'accès, qui est constitué du nom de fichier, précédé par le nom de répertoire qui le contient sous la forme :

chemin d'accès = répertoire / nom

◇ Arborescence

Principe classique de l'**arborescence** :

- une racine : désignée par "/" (en anglais «**slash**»)
- nœuds : répertoires
- feuilles : fichiers



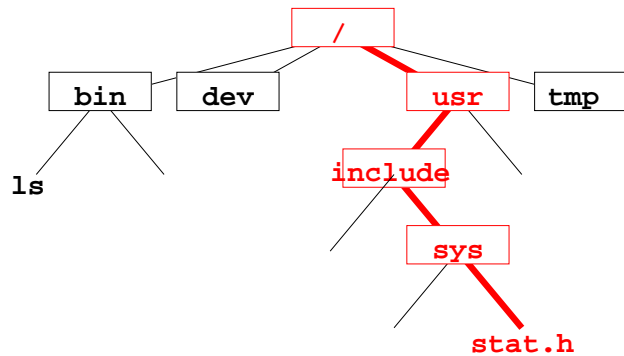
Selon ce schéma, 3 désignations possibles d'un fichier.

◇ chemin d'accès absolu :

chemin d'accès = répertoire / nom

Si le nom de répertoire commence par / , il s'agit d'une référence absolue par rapport au répertoire racine / , constituée d'une liste des répertoires à parcourir depuis la racine pour accéder au fichier.

par exemple : `/usr/include/sys/stat.h`



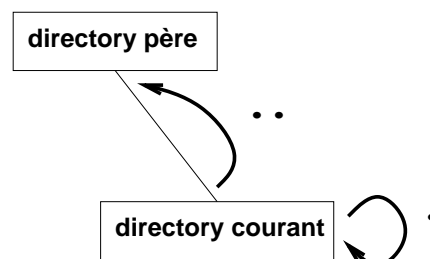
◇ chemin d'accès relatif :

chemin d'accès = répertoire / nom

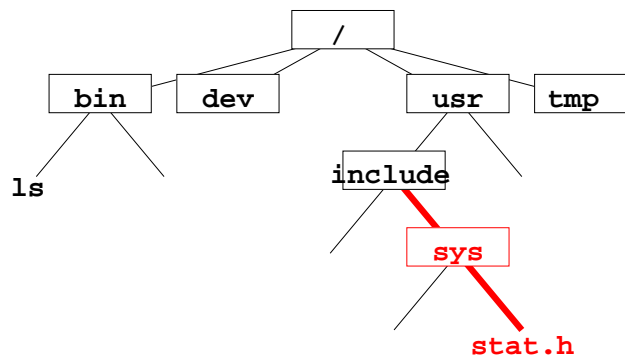
Si le nom de répertoire ne commence pas par / , il s'agit d'une référence relative par rapport au répertoire courant.

Le répertoire courant est noté `.`.

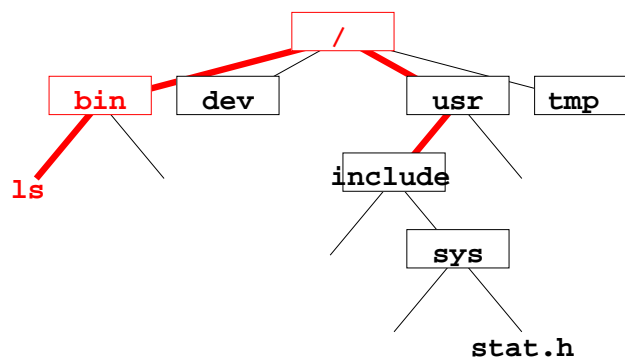
Le répertoire parent du répertoire courant est noté `..`.



par exemple, depuis `/usr/include/` : `sys/stat.h`



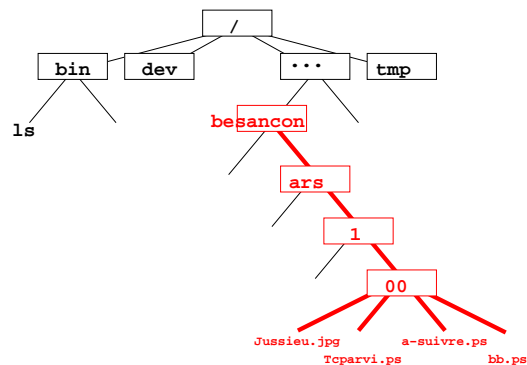
par exemple, depuis `/usr/include/` : `../../../../bin/ls`



◇ chemin d'accès relatif par rapport à un utilisateur :

Sur le principe identique à un chemin relatif sauf que le point de départ est le répertoire d'un utilisateur que l'on désigne par le signe `~` suivi du nom de l'utilisateur :

```
% ls ~besancon/ara/1/00
Jussieu.jpg  a-suivre.ps  Tcparvi.ps  bb.ps
```



§ 4.2 Rappel sur les options de commande Unix

Dans les transparents qui suivent, on se reportera quand nécessaire aux pages de manuel des commandes.

On rappelle que souvent les options des commandes peuvent être regroupées et permutées :

```
ls -a -l ≡ ls -al ≡ ls -la ≡ ls -l -a
```

§ 4.3 Liste des fichiers : *ls*

Commande à utiliser : *ls*

Nombreuses options :

```
% ls -z
```

```
ls: illegal option -- z
```

```
usage: ls [-lACFLRTacdfikloqrstu] [file ...]
```

Les options sont cumulables...

Les options les plus utiles :

- aucune option

affichage en colonne des noms des fichiers :

```
% ls
```

```
a          b          c          d          e          f
```

- option *-l*

affichage au format long des informations relatives aux fichiers :

```
% ls -l
```

```
total 2
```

```
drwxr-xr-x  2 besancon      512 Oct  1 16:42 a
-rw-r--r--  1 besancon         0 Oct  1 16:42 b
-rw-r--r--  1 besancon         0 Oct  1 16:42 c
-rw-r--r--  1 besancon         0 Oct  1 16:42 d
drwxr-xr-x  2 besancon      512 Oct  1 16:42 e
-rw-r--r--  1 besancon         0 Oct  1 16:42 f
```

- option -R

liste récursive à partir du répertoire courant :

```
% ls -R
a          b          c          d          e          f

a:
y          z

e:
t
```

- options -R plus -l

liste récursive avec plus de renseignements à partir du répertoire courant :

```
% ls -Rl
total 2
drwxr-xr-x  2 besancon    512 Oct  1 16:43 a
-rw-r--r--  1 besancon      0 Oct  1 16:42 b
-rw-r--r--  1 besancon      0 Oct  1 16:42 c
-rw-r--r--  1 besancon      0 Oct  1 16:42 d
drwxr-xr-x  2 besancon    512 Oct  1 16:43 e
-rw-r--r--  1 besancon      0 Oct  1 16:42 f

a:
total 0
-rw-r--r--  1 besancon      0 Oct  1 16:43 y
-rw-r--r--  1 besancon      0 Oct  1 16:43 z

e:
total 0
-rw-r--r--  1 besancon      0 Oct  1 16:43 t
```

- option -F

affichage des fichiers avec un suffixe désignant le type du fichier

```
% ls -F
a/      b      c      d      e/      f

% ls -lF
total 2
drwxr-xr-x  2 besancon      512 Oct  1 16:43 a/
-rw-r--r--  1 besancon           0 Oct  1 16:42 b
-rw-r--r--  1 besancon           0 Oct  1 16:42 c
-rw-r--r--  1 besancon           0 Oct  1 16:42 d
drwxr-xr-x  2 besancon      512 Oct  1 16:43 e/
-rw-r--r--  1 besancon           0 Oct  1 16:42 f
```

- option -a

affichage des fichiers dont les noms commencent par '.'

```
% ls -aF ~besancon
./              .lessrc        .tvwmrc
../             .login@         .twmrc
.Xauthority     .logout@        .weblink
.Xdefaults     .lynxrc         .workmandb
.Xresources    .mailcap@       .workmanrc
```

Par défaut, la commande *ls* n'affiche pas les noms de fichiers commençant par '.' qui par convention sont des fichiers de configuration d'utilitaires.

§ 4.4 Contenu d'un fichier texte : *cat*, *more*

Affichage du contenu d'un fichier (de préférence texte) :

```
cat [options] fichier1 fichier2 ... dernier-fichier
```

Par exemple :

```
% cat /etc/motd
```

```
SunOS Release 4.1.4 (EXCALIBUR [1.1]): Fri Aug 8 17:43:56 GMT 1997
```

```
This system is for the use of authorized users only. Individuals using
this computer system without authority, or in excess of their authority,
are subject to having all of their activities on this system monitored
and recorded by system personnel.
```

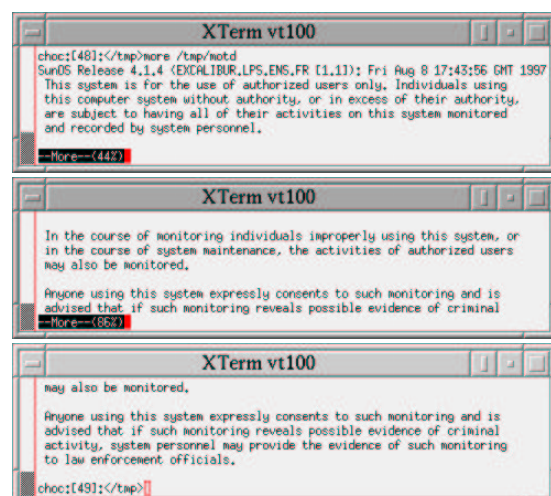
```
In the course of monitoring individuals improperly using this system, or
in the course of system maintenance, the activities of authorized users
may also be monitored.
```

```
Anyone using this system expressly consents to such monitoring and is
advised that if such monitoring reveals possible evidence of criminal
activity, system personnel may provide the evidence of such monitoring
to law enforcement officials.
```

En cas de texte très long, affichage page d'écran par page d'écran :

```
more [options] fichiers
```

- Caractère q pour quitter
- Caractère espace pour avancer d'une page d'écran
- Caractère b pour revenir en arrière d'une page (backward)
- Caractère f pour avancer d'une page d'écran (forward)



§ 4.5 Destruction d'un fichier : `rm`

Suppression par : `rm [options] fichiers`

Par cette commande, on efface des fichiers.

Quelques options :

"-i" : confirmation à chaque suppression

"-r" : suppression récursive

"-f" : suppression en force d'un fichier même si ses droits ne s'y prêtent pas

`rm -rf répertoires` permet de supprimer récursivement toute une arborescence sans demande de confirmation. Attention : dangereux.

```
% ls -F
a/      b      c      d      e/      f
```

```
% rm f
% ls -F
a/      b      c      d      e/
```

```
% rm a
rm: a is a directory
```

```
% rmdir a
rmdir: a: Directory not empty
% ls a
y      z
```

```
% rm -rf a
% ls -F
b      c      d      e/
```

```
% rm -i b
rm: remove b? y
% ls -F
c      d      e/
```

§ 4.6 Duplication d'un fichier : *cp*

Copie : *cp* [options] original destination

La destination peut être un fichier ou un répertoire. Dans le cas d'une destination répertoire, on déplace le fichier original en lui conservant son nom. Dans le cas d'une destination fichier, on renomme le fichier original.

quelques options :

"-i" : confirmation à chaque écrasement de fichier

"-r" : copie récursive

```
% ls -lF
total 3
drwxr-xr-x  2 besancon   512 Oct  1 16:43 a/
-rw-r--r--  1 besancon    0 Oct  1 16:42 b
-rw-r--r--  1 besancon    0 Oct  1 16:42 c
-rw-r--r--  1 besancon    0 Oct  1 16:42 d
drwxr-xr-x  2 besancon   512 Oct  1 16:43 e/
-rw-r--r--  1 besancon  342 Oct  1 17:25 f

% cp f g
% ls -lF
total 4
drwxr-xr-x  2 besancon   512 Oct  1 16:43 a/
-rw-r--r--  1 besancon    0 Oct  1 16:42 b
-rw-r--r--  1 besancon    0 Oct  1 16:42 c
-rw-r--r--  1 besancon    0 Oct  1 16:42 d
drwxr-xr-x  2 besancon   512 Oct  1 16:43 e/
-rw-r--r--  1 besancon  342 Oct  1 17:25 f
-rw-r--r--  1 besancon  342 Oct  1 17:25 g

% cp /tmp/motd h
% ls -l h
-rw-r--r--  1 besancon   752 Oct  1 17:26 h
```

§ 4.7 Déplacement et renommage d'un fichier : mv

Déplacement : `mv [options] original destination`

La destination peut être un fichier ou un répertoire.

Dans le cas d'une destination fichier, on **renomme** le fichier original.

```
% ls -lF
total 3
-rw-r--r-- 1 besancon      0 Oct  1 16:42 d
drwxr-xr-x 2 besancon    512 Oct  1 16:43 e
-rw-r--r-- 1 besancon    752 Oct  1 17:25 f

% mv f toto
% ls -lF
total 3
-rw-r--r-- 1 besancon      0 Oct  1 16:42 d
drwxr-xr-x 2 besancon    512 Oct  1 16:43 e/
-rw-r--r-- 1 besancon    752 Oct  1 17:25 toto
```

Dans le cas d'une destination répertoire, on **déplace** le fichier original en lui conservant son nom.

```
% mv toto /tmp
% ls -l toto
toto not found
% ls -l /tmp/toto
-rw-r--r-- 1 besancon    752 Oct  1 17:25 /tmp/toto
```

quelques options :

"-i" : confirmation à chaque écrasement de fichier

```
% mv -i b c
remove c? y
% ls -l b c
b not found
-rw-r--r-- 1 besancon    752 Oct  1 17:30 c
```

§ 4.8 Création de répertoires : `mkdir`

Création de répertoire : `mkdir` [options] répertoires

```
% mkdir z
% ls -lF
total 1
drwxr-xr-x  2 besancon      512 Oct  1 17:40 z/
```

Création directe de sous répertoires en cascade :

```
% mkdir -p repertoire1/ss-repertoire2/ss-ss-repertoire2
% ls -R
repertoire1/

repertoire1:
ss-repertoire2/

repertoire1/ss-repertoire2:
ss-ss-repertoire2/

repertoire1/ss-repertoire2/ss-ss-repertoire2:
```

§ 4.9 Suppression de répertoires : `rmdir`

Suppression de répertoire : `rmdir` [options] répertoires

```
% cp /etc/motd z/toto
% rmdir z
rmdir: z: Directory not empty
```

§ 4.10 Position dans les répertoires : *cd*, *pwd*

Changement de répertoire courant : *cd* répertoire

Affichage du répertoire courant : *pwd* (en anglais \equiv Present Working Directory)

```
% cd /etc
% pwd
/etc
% cd /usr/include
% pwd
/usr/include
% cd /inexistant
/inexistant: bad directory
```

Selon le shell, le message d'erreur dans le dernier cas peut changer :

```
% cd /inexistant
bash: /inexistant: No such file or directory
```

§ 4.11 Editeur de fichier texte – *vi*, *emacs*

Il existe beaucoup d'éditeurs de texte sous Unix mais seuls quelques uns sont suffisamment robustes pour être utilisés efficacement et avec confiance :

- Le seul éditeur de texte standard sous Unix : *vi*
(cf. <http://www.math.fu-berlin.de/~guckes/vi/> pour de la doc)
- *emacs*. Très puissant, complexe à maîtriser, simple une fois qu'on sait s'en servir. Cf <http://www.emacs.org>.
- *xemacs*. Variante plus graphique d'*emacs*.
- *nedit*. Simple et intuitif grâce à ses menus.

◇ vi

◇ emacs

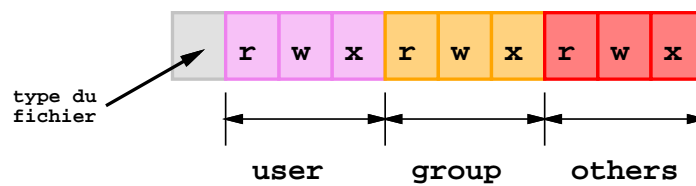
Chapitre 5 : Commandes auxiliaires de manipulation de fichiers Unix

§ 5.1 Droits d'accès : `chmod`

Les droits d'accès à un fichier se voient grâce à la commande `ls -l` :

```
% ls -l
total 16
-rw-r--r-- 1 besancon 15524 Sep 15 15:17 droits.idraw
```

Plus particulièrement, les droits sont indiqués par les 10 premiers caractères de chaque ligne affichée :



Il existe trois droits d'accès associés à chaque fichier :

- droits du propriétaire ($u \equiv \text{user}$)
- droits des membres du groupe ($g \equiv \text{group}$)
- droits des autres utilisateurs ($o \equiv \text{others}$)

Il existe trois types de permissions :

- droit en lecture ($r \equiv \text{read}$)
- droit en écriture ($w \equiv \text{write}$)
- droit en exécution ($x \equiv \text{execute access}$)

Pour changer les droits : `chmod [options] modes fichiers`

La précision des modes dans la commande peut prendre deux formes :

– forme symbolique :

"u", "g", "o" ou "a"

"+" ou "-" ou "="

permissions

– forme numérique :

Les permissions sont exprimées en base huit ou octale.

Par exemple : `rxwx rx-x rx-x` \equiv 755

Comment calculer en base octale ?

Droits	Valeur octale
---	0
--x	1
-w-	2
-wx	3
r--	4
r-x	5
rw-	6
rxw	7

C'est pourquoi on a par exemple :

`rxwx rx-x rx-x` \equiv 755

`rw- r-- r--` \equiv 644

`rw- --- ---` \equiv 600

etc.

```
% ls -l fichier
-rw-r--r--  1 besancon  software  249 Sep 20 22:43 fichier

% chmod g+w fichier
% ls -l fichier
-rw-rw-r--  1 besancon  software  249 Sep 20 22:43 fichier

% chmod o=rw fichier
% ls -l fichier
-rw-rw-rw-  1 besancon  software  249 Sep 20 22:43 fichier

% chmod 640 fichier
% ls -l fichier
-rw-r-----  1 besancon  software  249 Sep 20 22:43 fichier
```

Il existe d'autres modes spéciaux réservés à l'administrateur système :

- bit **setuid** (4000 en octal) : le programme est exécuté avec les droits de l'utilisateur propriétaire

```
% ls -lF fichier
-rwxr-xr-x  1 besancon  software      249 Sep 20 22:43 fichier*
% chmod u+s fichier
% ls -lF fichier
-rwsr-xr-x  1 besancon  software      249 Sep 20 22:43 fichier*
```

- bit **setgid** (2000 en octal) : le programme est exécuté avec les droits du groupe propriétaire

```
% chmod g+s fichier
% ls -lF fichier
-rwsr-sr-x  1 besancon  software      249 Sep 20 22:43 fichier*
```

§ 5.2 Recherche de chaînes de caractères : *grep*

Commande de recherche de chaîne de caractères dans un fichier :

```
grep [options] chaîne fichiers
```

quelques options :

"-i" : pas de différenciation entre lettres minuscules et majuscules

"-n" : affichage des numéros de ligne

"-l" : n'affiche que les noms de fichiers

"-v" : affichage des lignes ne contenant pas la chaîne précisée

Soit le fichier :

Ecrivons toto en minuscules ici.

Et ici ToTo en minuscules et majuscules.

Mais là on ne met pas la chaîne de l'exemple.

Quelques exemples d'utilisation de *grep* :

```
% grep toto fichier
```

Ecrivons toto en minuscules ici.

```
% grep -in toto fichier
```

1:Ecrivons toto en minuscules ici.

2:Et ici ToTo en minuscules et majuscules.

```
% grep -inv toto fichier
```

3:Mais là on ne met pas la chaîne de l'exemple.

Grep utilise des **regexp** (*regular expressions* \equiv *expressions rationnelles*) pour désigner de façon puissante des chaînes à rechercher :

`^` → désigne le début de ligne

`$` → désigne la fin de ligne

`.` → désigne un caractère quelconque

`*` → désigne une répétition de 0 à N fois du caractère précédent

`[caractères]` → désigne un caractère parmi la liste précisée

`[^caractères]` → désigne un caractère ne figurant pas parmi la liste précisée

§ 5.3 Modification à la volée de contenu de fichiers : *sed*

Commande d'édition de flux : `sed options fichiers`

On modifie un flux pas un contenu du fichier : après l'application de la commande, le fichier appliqué reste inchangé.

Par exemple : substitution à la volée de caractères

```
% cat fichier
```

```
Ecrivons toto ici.
```

```
Et ici ToTo.
```

```
Mais là on ne met pas la chaîne de l'exemple.
```

```
% sed -e 's/toto/XXXXX/g' fichier
```

```
Ecrivons XXXXX ici.
```

```
Et ici ToTo.
```

```
Mais là on ne met pas la chaîne de l'exemple.
```

Possibilité d'utiliser des regexp comme pour grep.

§ 5.4 Tri d'un fichier : *sort*

Commande de tri d'un fichier : *sort* [options] fichiers

Par exemple :

```
% cat fichier1
arbre
12
ascenseur
ordinateur
% sort fichier
12
arbre
ascenseur
ordinateur
```

§ 5.5 Comptage de caractères, mots, etc. : *wc*

Commande de comptage de caractères, de mots, de lignes dans un fichier :

wc [option] fichiers

quelques options :

"-c" : nombre de caractères uniquement

"-w" : nombre de mots uniquement

"-l" : nombre de lignes uniquement

Par exemple :

```
% wc fichier
      3      16      82 fichier
% wc -l fichier
      3 fichier
```

§ 5.6 Comparaison de fichiers : *diff*

Pour réaliser la comparaison du fichier texte `fichier2` par rapport au fichier texte `fichier1` : `diff [-c] fichier1 fichier2`

```
% diff fichier1 fichier2
1c1
< Deux fotes d'ortographe dans cette phrase.
---
> Deux fautes d'orthographe dans cette phrase.

% diff -c fichier1 fichier2
*** fichier1  Sun Sep  9 19:06:13 2001
--- fichier2  Sun Sep  9 19:06:24 2001
*****
*** 1 ****
! Deux fotes d'ortographe dans cette phrase.
--- 1 ----
! Deux fautes d'orthographe dans cette phrase.
```

§ 5.7 Manipulation des noms de fichiers : *basename*, *dirname*

Manipulation sur le nom du fichier : `basename fichier [suffixe]`

1. Suppression des composantes du path jusqu'au dernier caractère / :

```
% basename /users/adm/besancon/esigetel/cours.tex
cours.tex
```

2. La précision d'un suffixe amène la suppression de ce suffixe également :

```
% basename /users/adm/besancon/esigetel/cours.tex .tex
cours
```

Manipulation sur le chemin au fichier : `dirname fichier`

1. Suppression des composantes du path après le dernier caractère / :

```
% dirname /users/adm/besancon/esigetel/cours.tex  
/users/adm/besancon/esigetel
```
2. Dans le cas où il n'y a pas de caractère / :

```
% dirname cours.tex  
.
```

§ 5.8 Manipulation sur des lignes de fichiers : *head, tail*

- Extraction des premières lignes de fichier : `head [-N] fichiers`

Par exemple :

```
% head -3 /tmp/motd  
SunOS Release 4.1.4 (EXCALIBUR.LPS.ENS.FR [1.1]): Fri Aug 8 17:43:56 GMT 1997  
This system is for the use of authorized users only. Individuals using  
this computer system without authority, or in excess of their authority,
```

- Extraction des dernières lignes de fichier : `tail [-N] fichiers`
ou `tail [+N] fichiers`

Par exemple :

```
% tail -3 /tmp/motd  
advised that if such monitoring reveals possible evidence of criminal  
activity, system personnel may provide the evidence of such monitoring  
to law enforcement officials.
```

Par exemple :

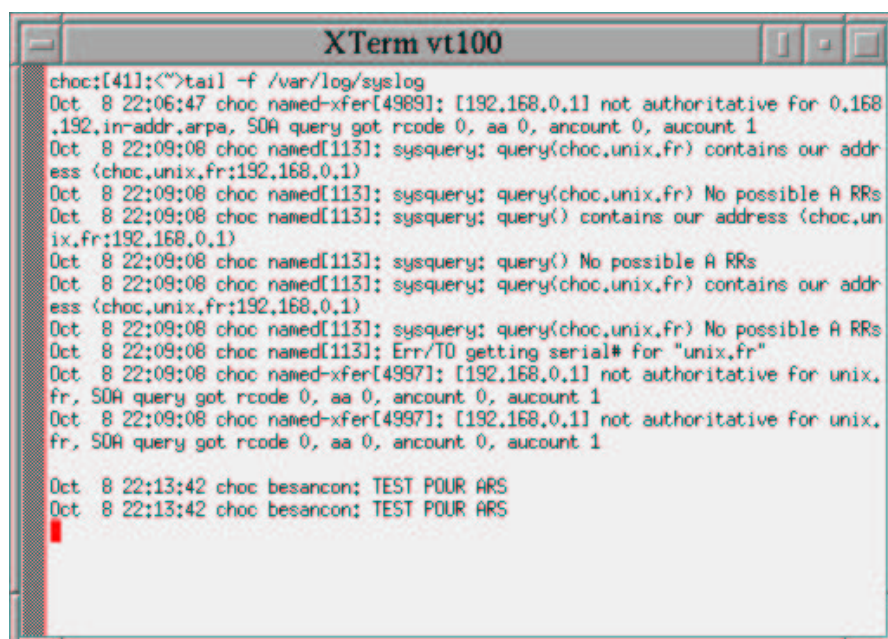
```
% tail +3 /tmp/motd
```

```
this computer system without authority, or in excess of their authority,  
are subject to having all of their activities on this system monitored  
and recorded by system personnel.
```

```
In the course of monitoring individuals improperly using this system, or  
in the course of system maintenance, the activities of authorized users  
may also be monitored.
```

```
Anyone using this system expressly consents to such monitoring and is  
advised that if such monitoring reveals possible evidence of criminal  
activity, system personnel may provide the evidence of such monitoring  
to law enforcement officials.
```

- Affichage des dernières lignes en temps réel : `tail -f fichier`



```
XTerm vt100  
choc:[41]:<~>tail -f /var/log/syslog  
Oct  8 22:06:47 choc named-xfer[4989]: [192.168.0.1] not authoritative for 0.168  
.192.in-addr.arpa, SOA query got rcode 0, aa 0, ancount 0, account 1  
Oct  8 22:09:08 choc named[113]: sysquery: query(choc.unix.fr) contains our addr  
ess <choc.unix.fr:192.168.0.1>  
Oct  8 22:09:08 choc named[113]: sysquery: query(choc.unix.fr) No possible A RRs  
Oct  8 22:09:08 choc named[113]: sysquery: query() contains our address <choc,un  
ix.fr:192.168.0.1>  
Oct  8 22:09:08 choc named[113]: sysquery: query() No possible A RRs  
Oct  8 22:09:08 choc named[113]: sysquery: query(choc.unix.fr) contains our addr  
ess <choc.unix.fr:192.168.0.1>  
Oct  8 22:09:08 choc named[113]: sysquery: query(choc.unix.fr) No possible A RRs  
Oct  8 22:09:08 choc named[113]: Err/TD getting serial# for "unix.fr"  
Oct  8 22:09:08 choc named-xfer[4997]: [192.168.0.1] not authoritative for unix.  
fr, SOA query got rcode 0, aa 0, ancount 0, account 1  
Oct  8 22:09:08 choc named-xfer[4997]: [192.168.0.1] not authoritative for unix.  
fr, SOA query got rcode 0, aa 0, ancount 0, account 1  
  
Oct  8 22:13:42 choc besancon: TEST POUR ARS  
Oct  8 22:13:42 choc besancon: TEST POUR ARS
```

§ 5.9 Recherche de fichiers : *find*

Commande de recherche de fichiers dans l'arborescence :

`find répertoire expressions`

On recherche à partir du répertoire indiqué les fichiers répondant aux critères exprimés par les expressions.

Les expressions indiquent :

- des conditions
- des actions à effectuer

Quelques expressions :

- critère de nom : `-name nom`
- critère de droits d'accès : `-perm permissions`
- critère de type (fichier, répertoire) : `-type type` (d pour directory, f pour file, etc.)
- critère de taille : `-size N`
- critère de date récente : `-newer fichier`
- critère de date : `-atime N, -mtime N, -ctime N`
- ou logique entre conditions : `condition1 -o condition2`
- et logique entre conditions : `condition1 -a condition2` (en fait le `-a` est facultatif)
- affichage du nom du fichier trouvé : `-print`
- exécution d'une commande : `-exec commande {} \;`

Par exemple :

- Recherche des fichiers d'extension '.c' :

```
% find . -name \*.c -print
```

- Recherche des répertoires :

```
% find . -type d -print
```

- Recherche des fichiers de plus de 1000000 caractères :

```
% find ~ -size +1000000c -print
```

- Recherche de tous les fichiers s'appelant a.out ou s'appelant avec une extension '.o', non utilisés depuis plus de 7 jours et on appliquera la commande d'effacement aux fichiers trouvés :

```
% find . \( -name a.out -o -name '*.o' \) -atime +7 -exec rm {} \;
```

§ 5.10 Compression de fichiers : *(un)compress, g(un)zip*

Quelques commandes de compression répandues :

- commande `compress fichier`

Le fichier compressé s'appelle dorénavant `fichier.Z`.

- commande `gzip fichier`

Le fichier compressé s'appelle dorénavant `fichier.gz`.

Cette commande compresse mieux les fichiers que `compress`

Pour décompresser un fichier et le ramener à sa taille originale :

- commande `uncompress fichier.Z`

Le fichier décompressé retrouve son nom `fichier`.

- commande `gunzip fichier.gz`

Le fichier décompressé retrouve son nom `fichier`.

§5.11 Archivage : tar

La commande `tar` permet d'archiver dans un seul fichier une arborescence. Conjuguée à une commande de compression, cela permet de mettre de côté une arborescence dont on n'a plus besoin. Au passage, on gagne de la place (cf chapitre 8 sur les quotas).

Selon l'action que l'on veut faire, la syntaxe est la suivante :

- Création d'une archive : `tar cvf archive.tar fichiers`
- Affichage du contenu d'une archive : `tar tvf archive.tar`
- Extraction de l'archive complète : `tar xvf archive.tar`
- Extraction d'un fichier précis de l'archive :
`tar xvf archive.tar fichier`

Vous pouvez selon les systèmes Unix compresser l'archive au fur et à mesure de sa construction :

- Utilisation de `compress` :
 - Création d'une archive : `tar cvZf archive.tar.Z fichiers`
 - Affichage du contenu d'une archive : `tar tvZf archive.tar.Z`
 - Extraction de l'archive complète : `tar xvZf archive.tar.Z`
 - Extraction d'un fichier précis de l'archive :
`tar xvZf archive.tar.Z fichier`
- Utilisation de `gzip` :
 - Création d'une archive : `tar cvzf archive.tar.gz fichiers`
 - Affichage du contenu d'une archive : `tar tvzf archive.tar.gz`
 - Extraction de l'archive complète : `tar xvzf archive.tar.gz`
 - Extraction d'un fichier précis de l'archive :
`tar xvzf archive.tar.gz fichier`

§5.12 Liens sur fichiers : `ln`, `ln -s`

A chaque fichier peuvent être associés plusieurs noms

Chaque nom est un **lien**.

Il y a un compteur de liens pour chaque fichier :

- incrémenté lors de la création d'un lien
- décrémenté lors de la suppression d'un lien
- le contenu d'un fichier est détruit lorsque le dernier lien est supprimé

Deux types de liens : lien **hard**, lien **symbolique**

- lien **hard** : limité au sein d'un même disque dur, où est garantie l'unicité d'un inode («matricule de fichier»)

La commande à utiliser est : `ln original synonyme`

Suppression par `rm`

```
% ls -l fichier1
-rw-r--r--  1 besancon  software  9919 Oct 17 18:25 fichier1

% ln fichier1 fichier2

% ls -l fichier1 fichier2
-rw-r--r--  2 besancon  software  9919 Oct 17 18:25 fichier1
-rw-r--r--  2 besancon  software  9919 Oct 17 18:25 fichier2

% ls -li fichier1 fichier2
689357 -rw-r--r--  2 besancon  software  9919 Oct 17 18:25 fichier1
689357 -rw-r--r--  2 besancon  software  9919 Oct 17 18:25 fichier2

% rm fichier1

% ls -li fichier2
689357 -rw-r--r--  1 besancon  software  9919 Oct 17 18:25 fichier2
```

- lien **symbolique** : non limité à un disque parce qu'utilisant le nom d'un fichier et non pas son «matricule»

En fait, c'est un fichier contenant le nom du fichier source.

La commande à utiliser est : `ln -s original synonyme`

Suppression par `rm`

```
% ls -l fichier1
-rw-r--r-- 1 besancon software 9919 Oct 17 18:25 fichier1

% ln -s fichier1 fichier2

% ls -liF fichier1 fichier2
689357 -rw-r--r-- 1 besancon software 9919 Oct 17 18:25 fichier1
689358 lrwxr-xr-x 1 besancon software 8 Oct 17 18:26 fichier2@ -> fichier1

% ls -lL fichier1 fichier2
-rw-r--r-- 1 besancon software 9919 Oct 17 18:25 fichier1
-rw-r--r-- 1 besancon software 9919 Oct 17 18:25 fichier2
```

```
% rm fichier1

% ls -liL fichier2
132845 lrwxrwxrwx 1 besancon software 8 Sep 20 22:29 fichier2@ -> fichier1

% cat fichier2
cat: fichier2: No such file or directory
```

§ 5.13 Langage plus complet : *awk*

C'est un utilitaire recherchant des motifs dans un fichier et réalisant des opérations sur les lignes répondant aux critères.

C'est plus généralement un mini langage de programmation à la syntaxe proche du langage C.

Il est très souvent utilisé pour réaliser des filtres sur des fichiers.

Se reporter au complément de cours sur *awk* :

`ftp://ftp.imag.fr/pub/DOC.UNIX/AWK/awk.pdf`

`ftp://ftp.imag.fr/pub/DOC.UNIX/AWK/awk.ps.gz`

Quelques exemples simples :

- Effacer le deuxième champ de chaque ligne du fichier `a.dat` :
`% awk '{$2= "" ; print}' a.dat > a.dat.new`
- Afficher le deuxième champ de chaque ligne du fichier `a.dat` :
`% awk '{print $2}' a.dat | more`
- Afficher toutes les lignes faisant plus de 72 caractères de long dans le fichier `a.dat` :
`% awk 'length > 72' a.dat`
- Tuer tous les processus Unix appartenant à l'utilisateur `thb` :
`% ps aux | awk '$1=="thb" {print "kill " $2}' | sh`

Un exemple plus compliqué montrant que l'on peut réaliser des opérations numériques ou des manipulations de chaînes sophistiquées :

Soit le fichier `program.awk` :

```
{
  x1 += $1
  x2 += $1*$1
}
END {
  x1 = x1/NR
  x2 = x2/NR
  sigma = sqrt(x2 - x1*x1)
  if (NR > 1) std_err = sigma/sqrt(NR - 1)
  print "Number of points = " NR
  print "Mean = " x1
  print "Standard Deviation = " sigma
  print "Standard Error = " std_err
}
```

On l'appelle sur le fichier `a.dat` par :

```
% awk -f program.awk a.dat
```

§ 5.14 Transfert de fichiers depuis et vers un lecteur de disquettes : *mcopy*

De vieux PC sous Unix permettent de transférer des fichiers depuis et vers leur lecteur de disquette. Le logiciel permettant d'utiliser les disquettes fonctionne sur la logique des commandes DOS.

La commande de base à utiliser est `mcopy`.

- Transfert depuis Unix vers la disquette : `mcopy fichier a:`
- Transfert de la disquette vers Unix : `mcopy a:fichier .`
- Affichage du contenu de la disquette : `mdir a:`

Se reporter à l'URL

<http://www.loria.fr/~giese/doc/mtools.html> pour plus de détails sur les commandes disponibles.

Chapitre 6 : Pratique du Bourne shell

§ 6.1 Principe d'exécution d'une commande

1. attente d'une entrée de commande ;
2. traitement des caractères spéciaux de la commande ;
3. recherche de l'exécutable ; s'il n'est pas trouvé, on affiche un message d'erreur et le shell reprend à l'étape 1 ;
4. `fork()` + `exec()` de la commande à lancer ;
5. le shell fait un `wait()` de la commande ;
6. une fois la commande terminée, le shell reprend à l'étape 1.

§ 6.2 Caractères spéciaux du shell : métacaractères

Caractères	Signification
tabulation, espace	appelés <i>white characters</i> ; délimiteurs des mots ; un tel caractère au minimum
retour charriot	fin de la commande à exécuter
&	lance une commande en tâche de fond
' " \	appelés <i>quote characters</i> ; changent la façon dont le shell interprète les caractères spéciaux
< > << >> `	caractères de redirection d'entrées/sorties
* ? [] [^]	caractères de substitution de noms de fichiers
\$	valeur d'une variable
;	séparateur de commandes sur une seule ligne

§ 6.3 Contrôle des commandes lancées : `&`, `fg`, `bg`, `kill`, `^C`, `^Z`

◇ Avant plan

Lorsqu'une commande est en train de s'exécuter, le shell ne rend pas la main et attend que la commande se termine (correctement ou incorrectement).

Pour interrompre prématurément une commande : taper sur la touche `Control` **et** **aussi** sur la touche `C` du clavier. Cela tue la commande qui tournait.

On notera l'appui sur ces 2 touches par `Ctrl C` ou par `^C`.

◇ Arrière plan

Si l'on veut une lancer une commande et récupérer la main tout de suite, avant même que la commande ait fini de s'exécuter, il faut lancer la commande par :

```
% commande &
```

Le signe `&` signifie de lancer en **tâche de fond**, en **background** la commande.

Sans ce signe, la commande est lancée en **premier plan**, en **foreground**.

◇ Passage en arrière plan

Pour passer en background une commande lancée en foreground :

1. Figurer la commande en cours

Taper sur la touche `Control` **et aussi** sur la touche `Z`, soit `Ctrl-Z` ou `^Z` :

```
% commande
```

```
...
```

```
^Z
```

```
[1]+  Stopped                  commande
```

2. Indiquer de l'exécuter dorénavant en background

Taper la commande **bg** :

```
% bg
```

```
[1]+  commande &
```

D'une façon générale, un débutant Unix doit proscrire l'utilisation de `Ctrl Z`.

Dans 9 cas sur 10, c'est `Ctrl C` qu'il doit employer. On évitera ainsi une saturation de la machine avec des commandes suspendues et en attente d'être tuées ou relancées.

◇ Passage au premier plan

Pour passer en foreground une commande lancée en background :

1. La commande est lancée

On a la main :

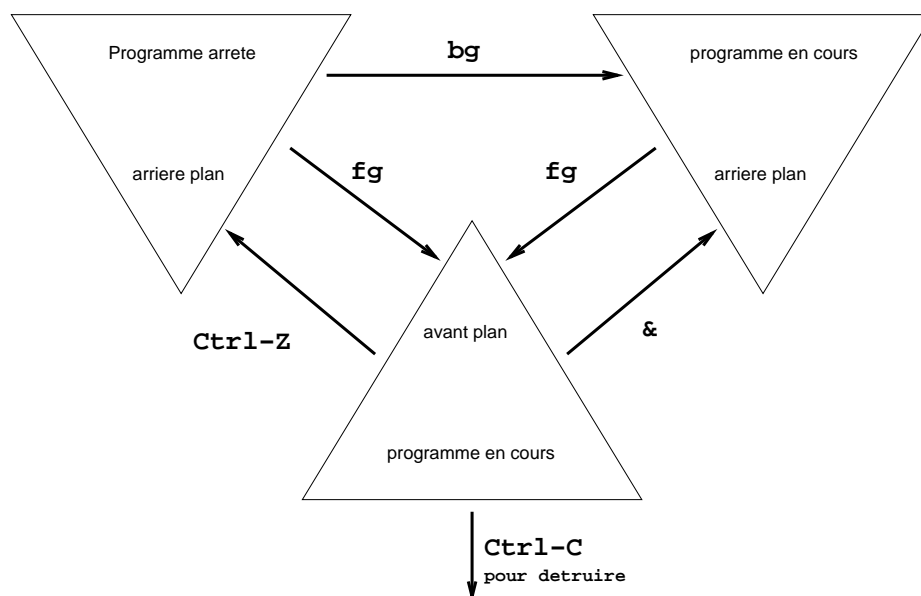
```
% commande &
```

...

2. Indiquer de l'exécuter dorénavant en foreground

Taper la commande **fg** :

```
% fg
```

◇ En résumé

◇ Liste des processus en arrière plan

Pour connaître la liste des commandes en background :

```
% jobs
```

```
[1]+  Running                  commande
```

On peut avoir plusieurs commandes en background. D'où une numérotation des commandes qui sont affichées. Ce numéro peut être repris dans les commandes *fg* et *bg* ainsi que dans la commande suivante, *kill*.

◇ Tuer un processus en arrière plan

Pour tuer une commande en background :

```
% jobs
```

```
[1]+  Running                  commande
```

```
% kill %1
```

```
[1]+  Terminated              commande
```

§ 6.4 Contrôle des processus : *ps*, *kill*, *top*

Les commandes *fg*, *bg*, *jobs* ne fonctionnent que sur les processus lancés par le shell courant. Les commandes vues précédemment peuvent donc être inutilisables si vous avez quitté votre shell.

◇ Commande *ps*

La commande *ps* plus générale permet d'avoir des informations sur tous les processus de la machine.

2 syntaxes selon l'Unix de la machine :

- Syntaxe de la famille d'Unix BSD
 - les processus associés à son terminal : *ps*
 - tous ses processus : *ps -x*
 - tous les processus de la machine : *ps -ax*
 - tous les processus de la machine avec les noms de login associés :
ps -aux

Exemple (partiel) de *ps -aux* :

```
% ps -aux
USER      PID  %CPU  %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1   0.0   0.1  1120    52 ?        S    Oct23   0:06  init
root         2   0.0   0.0     0     0 ?        SW   Oct23   0:00  [kflushd]
root         3   0.0   0.0     0     0 ?        SW   Oct23   0:01  [kupdate]
...
nobody    476   0.0   0.1  1300    44 ?        S    Oct23   0:01  [identd]
daemon    490   0.0   0.0  1144     0 ?        SW   Oct23   0:00  [atd]
xfs        636   0.0   0.3  2820   120 ?        S    Oct23   0:18  xfs -droppriv -da
root    14703   0.0   0.0  2256     0 tty1    SW   Oct25   0:00  [login]
root     9813   0.0   0.0  6912     0 ?        SW   Oct31   0:09  [kdm]
idiri    20810   0.0   0.0  6552     0 ?        SW   15:13   0:01  [kwm]
idiri    20837   0.0   0.0  2080     0 ?        SW   15:13   0:00  [tcsh]
idiri    20863   0.0   0.0  1996     0 pts/0   SW   15:13   0:00  [tcsh]
besancon 21785   0.0   1.3  1732   416 pts/1   S    15:25   0:00  -bash
idiri    23600   0.0   0.0  1844     0 tty1    SW   16:26   0:00  [vi]
idiri    23660   0.2   1.5  1860   472 tty2    S    16:39   0:01  vi probleme6.c
...
```

- Syntaxe de la famille d'Unix System-V
 - les processus associés à son terminal : `ps`
 - tous les processus de la machine avec les noms de login associés : `ps -edf`

Exemple (partiel) de `ps -edf` :

```
% ps -edf
  UID    PID  PPID  C   STIME TTY      TIME CMD
  root      0      0  0  09:09:47 ?        0:01 sched
  root      1      0  0  09:09:47 ?        0:02 /etc/init -
  root      2      0  0  09:09:47 ?        0:00 pageout
  root      3      0  0  09:09:47 ?        0:52 fsflush
  root    181      1  0  09:12:07 ?        0:06 /usr/lib/autofs/automountd
...
  daemon   283      1  0  09:12:12 ?        0:11 /usr/sbin/lpd
  root     291      1  0  09:12:13 ?        0:00 /usr/local/apache/bin/httpd
  root     296      1  0  09:12:14 ?        0:00 /usr/local/admin/lib/idled
  nobody  15130    291  0  23:30:56 ?        0:00 /usr/local/apache/bin/httpd
  besancon 16463 16461  0  00:12:26 pts/0    0:00 -csh
...
```

◊ Commande `kill`

La commande **kill** sert à communiquer avec des processus :

- arrêt de processus
- demande au processus de se reconfigurer
- passage en mode verbeux du processus
- etc.

La commande **kill** existe sur tous les Unix et il n'y a pas de différence de fonctionnement selon les Unix.

Syntaxe :

NAME

kill - terminate or signal a process

SYNOPSIS

kill [-s signal_name] pid ...

kill -l [exit_status]

kill -signal_name pid ...

kill -signal_number pid ...

Avec :

Some of the more commonly used signals:

1	HUP (hang up)
2	INT (interrupt)
3	QUIT (quit)
6	ABRT (abort)
9	KILL (non-catchable, non-ignorable kill)
14	ALRM (alarm clock)
15	TERM (software termination signal)

Par exemple, les 2 formes suivantes sont équivalentes :

```
kill -9 2878
```

```
kill -KILL 2878
```

Les signaux les plus utiles sont :

– SIGHUP

Cela envoie l'équivalent du Ctrl C du clavier.

– SIGKILL

Cela envoie un signal que le processus est obligé de suivre et qui se traduira inélectablement par la mort du processus.

◊ Commande *top*

Inconvénient de *ps* : c'est la liste des processus à un instant *t*.

On ne pourra jamais sous Unix avoir la liste des processus en cours : «principe de Heidelberg», le temps de chercher les processus et de faire le rapport, certains processus peuvent avoir disparu.

Amélioration de *ps* : la commande "*top*" qui n'est cependant pas standard sur tous les Unix.

Son intérêt : elle affiche une liste des processus toutes les *n* secondes

```

XTerm vt100
last pid: 21509; load averages: 0.02, 0.03, 0.04 14:52:22
71 processes: 69 sleeping, 1 running, 1 on cpu
CPU states: 98.2% idle, 0.6% user, 0.6% kernel, 0.6% iowait, 0.0% swap
Memory: 128M real, 141M swap in use, 445M swap free

  PID USERNAME  THR PRI NICE  SIZE  RES STATE  TIME  CPU COMMAND
21507 thb      1  58   0 2544K 1592K cpu    0:00 0.73% top
   326 thb      1  48   0   28M   17M run   77:44 0.52% Xsun
21486 thb      1  48   0 2576K 1904K sleep 0:00 0.24% bash
21485 thb      1  58   0 4120K 3128K sleep 0:00 0.14% xterm
21509 thb      1  58   0 2920K 1616K sleep 0:00 0.14% xwd
18503 thb      1  58   0 4128K 2760K sleep 0:06 0.11% xterm
   327 thb      1  58   0 2704K 1432K sleep 1:29 0.05% fvwm
18504 thb      1  48   0 2592K 1736K sleep 0:02 0.03% bash
   539 thb      1  59   0   48M   14M sleep 24:41 0.01% emacs-20.4
   362 thb      1  13  15 3360K 1560K sleep 2:04 0.00% xbuffy
20528 thb      1  59   0   47M   25M sleep 5:46 0.00% netscape
   215 root     10  51   0 3040K 1880K sleep 0:50 0.00% nsd
     1 root      1  58   0   776K  144K sleep 0:31 0.00% init
   275 root      1  58   0 1896K  552K sleep 0:22 0.00% ssfd1
   133 root      1  58   0 1856K  696K sleep 0:10 0.00% in.ndpd
  
```

§ 6.5 Quote characters : ' , " , \

Caractères	Nom	Description
'	single quote	le shell n'interprète aucun caractère spécial entre deux '
"	double quote	le shell n'interprète aucun caractère spécial à l'exception de \$ ` et \
\	backslash	le shell n'interprète pas le caractère spécial suivant le backslash

Exemples :

1. # echo "\$HOME"
/root
2. # echo '\$HOME'
\$HOME
3. # echo "\\$HOME"
\$HOME
4. # echo '\\$HOME'
\\$HOME
5. % echo fichier = ; ls
fichier =
fichier1 fichier2
6. % echo fichier = \; ls
fichier = ; ls

§ 6.6 Caractères de redirection : <, >, >>, <<, '|', 2>, >&

Toutes les entrées/sorties d'Unix sont réalisées au moyen de fichiers.

Chaque processus ouvre donc un certain nombre de fichiers. Ces fichiers sont référencés en interne par une table d'entiers dits **file descriptors**.

Nom	File descriptor	Destination par défaut
standard input (stdin)	0	clavier
standard output (stdout)	1	écran
standard error (stderr)	2	écran

Les file descriptors existent en langage C et sont profondément ancrés dans le fonctionnement même d'UNIX.

◇ stdin

Les fichiers système de programmation C indiquent :

```
% grep stdin /usr/include/stdio.h
#define stdin    (&__sF[0])
#define getchar()      getc(stdin)
```

ou cf le petit programme C suivant :

```
#include<stdio.h>
main( )
{
    char line[1024];
    (void)fgets(line, 1024, stdin);
}
```

◇ stdout

Les fichiers système de programmation C indiquent :

```
% grep stdout /usr/include/stdio.h
#define stdout  (&__sF[1])
#define putchar(x)          putc(x, stdout)
```

ou cf le petit programme C suivant :

```
#include<stdio.h>
main( )
{
    fprintf(stdout, "Bonjour !\n");
}
```

◇ stderr

Les fichiers système de programmation C indiquent :

```
% grep stderr /usr/include/stdio.h
#define stderr  (&__sF[2])
```

ou cf le petit programme C suivant :

```
#include<stdio.h>
main( )
{
    fprintf(stderr, "Enfer et damnation !\n");
}
```

Format	Description
<code>command < fichier</code>	stdin de <code>command</code> provient de <code>fichier</code>
<code>command > fichier</code>	stdout de <code>command</code> placé dans <code>fichier</code> dont le contenu précédent est écrasé
<code>command >> fichier</code>	stdout de <code>command</code> placé en fin de <code>fichier</code>
<code>command << label</code>	stdin de <code>command</code> provient des lignes de commande suivantes jusqu'à la ligne ne contenant que <code>label</code>
<code>'command'</code>	remplace <code>'command'</code> par le résultat de l'exécution de <code>command</code>
<code>command1 command2</code>	passer le stdout de <code>command1</code> comme stdin de <code>command2</code>
<code>command 2> fichier</code>	redirige stderr de <code>command</code> dans <code>fichier</code>
<code>command >& file-descriptor</code>	redirige la sortie de <code>command</code> vers un certain fichier <code>descriptor</code>

Exemples :

1. `% ls > /tmp/foo`

2. `% ls /etc >> /tmp/foo`

3. `% cat <<EOF > /dev/null`
`>>jghsfdgkfdhgkdjh`
`>>gd`
`>>EOF`

```
4. % n='wc -l /etc/passwd'
    % echo $n
    170 /etc/passwd

    % echo `wc -l /etc/passwd`
    170 /etc/passwd

5. % cat /etc/group | more

    qui équivaut à

    more /etc/group
```

```
6. % ls xx
    ls: xx: No such file or directory

    % ls xx > errors
    ls: xx: No such file or directory

    % ls xx 2> errors
    % cat errors
    ls: xx: No such file or directory
```

```
7. % ls cours.tex xx >&2 2> errors  
    cours.tex
```

```
% ls cours.tex xx 2> errors >& 2  
%
```

Quelques explications :

Le shell évalue la ligne de commande de gauche à droite.

Dans la commande "ls cours.tex xx >&2 2> errors", stdin est redirigé sur l'écran puisque stdout équivaut à l'écran à cet instant, puis stdout est redirigé sur errors. D'où le résultat.

◇ Résumé lego

- Entrée d'une commande (filedescriptor 0) :



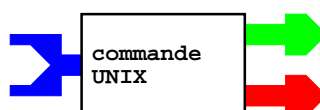
- Sortie d'une commande (filedescriptor 1) :



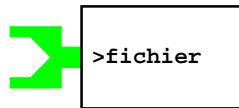
- Sortie d'erreur d'une commande (filedescriptor 2) :



- Commande type :



- Création d'un fichier contenant le flux de sortie de la commande :



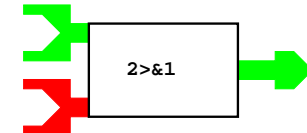
- Connexion de deux commandes :



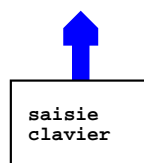
- Contrôle de l'erreur :



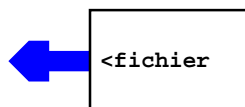
- Fusion de la sortie d'erreur et de la sortie de la commande



- Alimentation de la commande via l'entrée normale :

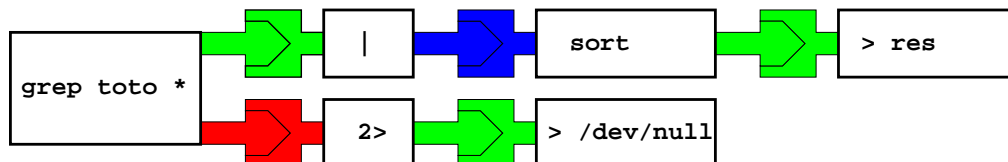


- Alimentation de la commande via un fichier :

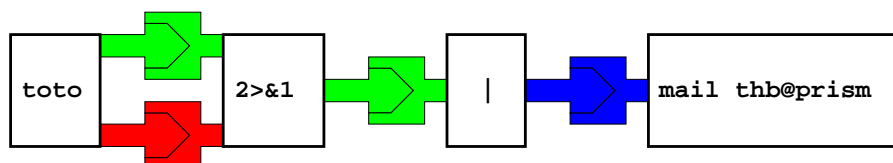


Deux exemples avec ces symboles :

```
- grep toto * 2>/dev/null | sort > res
```



```
- toto 2>&1 | mail thb@prism
```



§ 6.7 Désignation des fichiers par leurs noms : *, ?, [], [^]

Caractère	Description
*	0 ou plus caractères
?	1 caractère exactement
[]	1 caractère dans l'ensemble entre crochets
[^]	1 caractère non énuméré dans l'ensemble entre crochets

Exemples :

1. % `ls *`
`cours.tex`
2. % `ls /etc/*.??`
`/etc/locate.rc /etc/pwd.db /etc/spwd.db`
`/etc/mail.rc /etc/sendmail.cf`
3. % `ls /var/log/[lp]*`
`/var/log/lastlog /var/log/lpd-errs /var/log/ppp.log`
4. % `ls /var/log/[^mw]*`
`/var/log/dmesg /var/log/ppp.log`
`/var/log/dmesg.today /var/log/sendmail.st`
`/var/log/dmesg.yesterday /var/log/setuid.today`
`/var/log/lastlog /var/log/setuid.yesterday`
`/var/log/lpd-errs /var/log/slip.log`

§ 6.8 Variables

A un shell est associé un environnement dans lequel sont affectées des variables.

– **Assignation de variable**

`variable=valeur`

Rappel : l'espace est un caractère spécial donc pas d'espaces de part et d'autre du signe =

– **Consultation de variable**

`$variable` ou `${variable}`

– **Suppression de variable**

`unset variable`

– **Liste des variables définies dans l'environnement**

`set`

– **Protection en écriture**

`readonly variable`

`readonly`

– Variables de type numérique

Ca n'existe pas en Bourne shell. Les variables sont de type alphanumériques.

⇒ Il est donc impossible de faire :

```
count=1
count=$count + 1
```

La bonne façon de faire est d'utiliser la commande Unix **expr** :

```
count=1
count=`expr $count + 1`
```

Se reporter à la page de manuel de **expr**.

– Substitution avancée de variables

Variable	Description
<code>\${variable:-word}</code>	remplace par la valeur de variable si elle a une valeur ; sinon remplace par word ; n'affecte pas word à variable !
<code>\${variable:=word}</code>	Comme au dessus sauf qu'il y a affectation si variable n'est pas définie
<code>\${variable:?word}</code>	remplace par la valeur de variable si elle a une valeur ; sinon affiche word sur stderr ; si word est absent, affiche : "parameter null or not set"
<code>\${variable:+word}</code>	remplace la valeur de variable si elle a une valeur par word ; sinon ne fait rien

§ 6.9 Variables d'environnement

```
% foo=3
% sh
    % echo $foo

    % exit
% echo $foo
3
```

⇒ Les variables ne sont pas héritées par défaut par les commandes lancées par le shell.

La solution consiste à utiliser **export** (son pendant est bien sûr **unexport**) :

```
% FOO=33
% export FOO
% sh
    % echo $FOO
    33
    % exit
% echo $FOO
33
```

NB : traditionnellement, les variables d'environnement sont écrites en lettres majuscules

ATTENTION!!!:

```
% FOO=33
% export FOO
% sh
    % echo $FOO
    33
    % FOO=32
    % echo $FOO
    32
    % exit
% echo $FOO
33
```

L'environnement est hérité en accès en lecture uniquement.

Voici la liste des variables d'environnement standard :

Variable	Description
HOME	homedirectory
USER	username
SHELL	path du shell utilisé
PATH	liste des directories dans lesquels chercher des commandes
TERM	type du terminal utilisé

Les autres variables d'environnement ne sont pas standard.

§ 6.10 Ordre d'évaluation de la ligne de commande

1. Redirection des entrées/sorties
2. Substitution des variables
3. Substitution des noms de fichiers

Exemples :

```
1. % pipe=\  
   % echo $pipe  
   |
```

Explications :

- (a) pas de caractères de redirection des entrées/sorties ;
la commande est "echo \$pipe"
- (b) remplacement de la variable par son contenu ;
la commande est "echo |"
- (c) pas de caractères de substitution de noms de fichiers ;
la commande est "echo |"

```
2. % star=\*
    % echo $star
    cours.aux cours.dvi cours.log cours.ps cours.tex
```

Explications :

- (a) pas de caractères de redirection des entrées/sorties ;
la commande est "echo \$star"
- (b) remplacement de la variable par son contenu ;
la commande est "echo *"
- (c) remplacement du caractère * par la liste des fichiers ;
la commande est
"echo cours.aux cours.dvi cours.log cours.ps cours.tex"

§ 6.11 Double évaluation

On peut souhaiter évaluer une commande deux fois. Pour cela, on utilise **eval** :

```
% premier=TERM
% second=premier
% echo $second
premier
% eval echo \$$second
TERM
```

Explications :

1. D'après l'ordre d'évaluation,
 - (a) pas de caractères de redirection des entrées/sorties ;
la commande est "eval echo \\$\$second"
 - (b) remplacement de la variable par son contenu ;
la commande est "eval echo \\$\$premier"
 - (c) pas de caractères de substitution de noms de fichiers ;
la commande est "eval echo \\$\$premier"
2. Le caractère \$ perd son caractère spécial à cause du \ qui le précède ; la commande est "eval echo \$premier"
3. eval interprète ses arguments "echo \$premier" selon les règles du shell ce qui donne TERM.

§6.12 Quitter un shell : exit, Ctrl-D

Pour finir : taper exit ou Ctrl-D

Chapitre 7 : Programmation en Bourne shell

Le shell propose un langage de programmation interprété.

Son utilité :

- automatisation d'actions
- utilisation de structures plus avancées :
 - boucles
 - tests
 - ...
- scripts d'installation de logiciels à adapter

§ 7.1 Caractéristiques d'un shell script

Caractéristiques :

- C'est un programme écrit en langage shell.
- Il est écrit pour un shell particulier, à la syntaxe bien particulière. Un shell script ne peut pas être exécuté par un autre shell en général.
- Il est exécutable :

```
% file /usr/bin/true
/usr/bin/true: executable shell script
% ls -lF /usr/bin/true
-rwxr-xr-x 1 root staff 63 Oct 14 1994 /usr/bin/true*
```

§ 7.2 Structure d'un shell script

Désignation du shell utilisé

La première ligne du shell script commence par # ! suivi du path du shell utilisé et de ses arguments éventuels.

Commentaires

Un commentaire est introduit par le caractère # et se poursuit jusqu'à la fin de la ligne.

Un commentaire peut être placé n'importe où.

La première ligne du script est un commentaire très particulier.

Code

Traditionnelles lignes de code respectant la syntaxe du shell utilisé.

Exemple

```
% cat /usr/bin/true
#! /bin/sh
#
#      @(#)true.sh 1.5 88/02/07 SMI; from UCB
#
exit 0
```

ATTENTION!!!

Ceci est faux (pas uniquement à cause de l'orthographe) :

```
#####
#
# ARS 1999/2000
# AUTEUR : BERGOUGNOUX YVES
# DATE DE CREATION : 07/01/2000
# DATE DE MODIFICATION : 07/01/2000
# THEME : enrgitre l'expiration d'un compte de stagiaire ayant les memes #
# droit que sont parain pour géré la fin du compte dans le fichier cpt_sta#
# NOM DE LA COMMANDE : hda5/home/yves/projet/set-deadline (disque UNIX) #
#
#####

#!/bin/sh

...
```

Pourquoi ?

A cause de la position de la ligne `#!/bin/sh`

En l'absence d'indication de l'interpréteur de commandes en première ligne du script, le script est exécuté par le shell courant de la session de l'utilisateur.

Attention aux systèmes comme Linux où le shell par défaut est compatible avec le Bourne shell, masquant ainsi l'erreur!!!

Preuve :

Soit le script suivant que j'appellerai "script-foireux" :

```
# Je suis un commentaire qui n'a rien a faire ici

#!/bin/sh

for i in *
do
    echo $i
done
```

Si l'on exécute ce script, on obtient selon le shell de la session :

<pre>% echo \$SHELL /bin/csh % ./script-foireux for: Command not found. do: Command not found. i: Undefined variable.</pre>	<pre>% echo \$SHELL /bin/bash % ./script-foireux a repertoire1 script-foireux</pre>
------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------

**LA PREMIERE LIGNE DU SCRIPT DOIT ETRE CELLE
EN #!/bin/sh**

§ 7.3 Passage de paramètres à un shell script : \$1 à \$9

Comme tout programme, on peut passer des paramètres à un shell script.

Variable	Description
\$0	Nom du shell script
\$1 à \$9	Les 9 premiers paramètres
\$#	Le nombre de paramètres
\$*	Tous les paramètres passés au shell script sous la forme de mots individuels séparés
\$@	Tous les paramètres passés au shell script

Exemple :

```
% cat foo
#!/bin/sh
VAL=`expr ${1:-0} + ${2:-0} + ${3:-0}`
echo $VAL

% ./foo 12 3 8
23
% ./foo 5 7
12
% ./foo 13
13
```

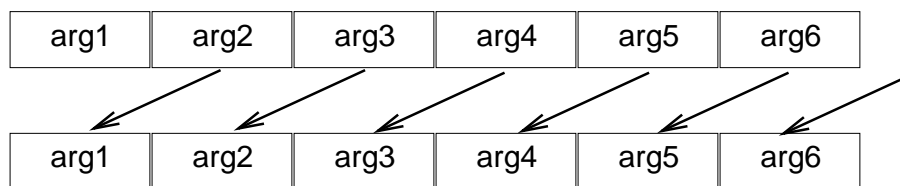
Comment accéder à tous les paramètres ?

```
% cat wrong
#!/bin/sh
echo $0 $1 $2 $3 $4 $5 $6 $7 $8 $9 $10 $11 $12

% ./wrong a b c d e f g h i j k l
./wrong a b c d e f g h i a0 a1 a2
```

⇒ comment faire ?

La solution consiste à utiliser la commande **shift** :



```
% cat good
#!/bin/sh
echo $1 $2 $3
shift
echo $1 $2 $3

% ./good a b c d
a b c
b c d
```

Attention !

A chaque emploi de `shift`, le paramètre `$1` précédent est perdu. Du même coup, ce paramètre est supprimé de `$*` et `$@`, `$#` est décrémenté de 1.

<code>#!/bin/sh</code>	<code>% ./show a b c d</code>
<code>echo "\$# ; \$1 \$2 \$3 ; \$@"</code>	<code>4 ; a b c ; a b c d</code>
<code>shift</code>	<code>3 ; b c d ; b c d</code>
<code>echo "\$# ; \$1 \$2 \$3 ; \$@"</code>	<code>2 ; c d ; c d</code>
<code>shift</code>	<code>1 ; d ; d</code>
<code>echo "\$# ; \$1 \$2 \$3 ; \$@"</code>	<code>0 ; ;</code>
<code>shift</code>	
<code>echo "\$# ; \$1 \$2 \$3 ; \$@"</code>	
<code>shift</code>	
<code>echo "\$# ; \$1 \$2 \$3 ; \$@"</code>	

Attention !

L'emploi de `shift` nécessite que le shell script ait au moins un paramètre :

<code>% cat good</code>	<code>% ./good <pas de paramètres></code>
<code>#!/bin/sh</code>	
<code>echo \$1 \$2 \$3</code>	<code>shift: can't shift that many</code>
<code>shift</code>	
<code>echo \$1 \$2 \$3</code>	

§ 7.4 Liste des paramètres d'un shell script : \$*, \$@

\$*	Tous les paramètres passés au shell script sous la forme de mots individuels séparés
\$@	Tous les paramètres passés au shell script

Soit un shell script à qui on passe :

```
% ./foo "arg1" "deux mots" "arg3"
```

Alors :

\$* est une liste de **4** éléments :

1. "arg1"
2. "deux"
3. "mots"
4. "arg3"

\$@ est une liste de **3** éléments :

1. "arg1"
2. "deux mots"
3. "arg3"

§ 7.5 Variables prédéfinies : \$?, \$!, \$\$

\$?

Contient le code de retour de la dernière commande exécutée.

\$!

Contient le PID de la dernière commande lancée en tâche de fond depuis le shell script.

\$\$

Contient le PID du shell script qui est en train de s'exécuter.

On ne peut que consulter ces variables.

§ 7.6 Commandes internes du shell : *builtins*, *type*

Le shell dispose de commandes internes (dites aussi *builtins*) : *cd*, *set*...

Une méthode pour les identifier est d'utiliser la commande du Bourne Shell **type** (*which* en C shell).

```
% type cd
cd is a shell builtin
% type echo
echo is a shell builtin
% type ls
ls is /bin/ls
```

§ 7.7 Commande d'affichage : *echo*

La commande d'affichage de caractères est **echo**.

Attention !

La commande *echo* peut ne pas être un builtin du shell. En Bourne Shell, c'est toujours le cas.

Ainsi souvent il existe une vraie commande Unix *echo* :

```
% ls -l /bin/echo
-r-xr-xr-x 1 bin  bin  32768 May 20 12:30 /bin/echo
```

Le comportement du builtin peut être différent de celui de la commande Unix.

La commande `echo` comprend des séquences semblables à celles de `printf()` :

Séquence	Description
<code>\b</code>	Backspace
<code>\c</code>	Pas de newline envoyé
<code>\n</code>	Newline
<code>\r</code>	Carriage return
<code>\t</code>	Tabulation horizontale
<code>\v</code>	Tabulation verticale
<code>\\</code>	Backslash
<code>\nnn</code>	Caractère dont le code octal ASCII est donné

Exemples :

```
1. #!/bin/sh
   echo -n "Il y a "
   n=`ls | wc -l`
   echo "$n fichiers ici"

2. #!/bin/sh
   echo "Il y a \c"
   n=`ls | wc -l`
   echo "$n fichiers ici"
```

§ 7.8 Entrée interactive : `read`

La commande **read** permet de lire au clavier et de placer les mots lus dans une liste de variables.

Syntaxe :

```
read variable-list
```

Le premier mot va dans la première variable, le deuxième mot va dans la deuxième variable... Tous les mots en trop sont stockés dans la dernière variable mentionnée.

Exemples :

```
1. % cat foo
#!/bin/sh
read var1 var2
echo "Premier mot : $var1"
echo "Le reste      : $var2"

% ./foo
Unix MS-DOS Windows 95 Windows NT MacOS
Premier mot : Unix
Le reste      : MS-DOS Windows 95 Windows NT MacOS
```

```
2. % cat foo
    #! /bin/sh -
    while read line
    do
        echo "$line"
    done

    % ./foo < /etc/motd
    FreeBSD 2.2.2-RELEASE (TOURNESOL.LPS.ENS.FR) #3: ...

    Welcome to FreeBSD!
    %
```

§7.9 Structure *if - then - else*

Syntaxes :

1. <i>if</i> condition	2. <i>if</i> condition	3. <i>if</i> condition
then	then	then
commands1	commands1	commands1
<i>fi</i>	else	elif condition2
	commands2	commands2
	<i>fi</i>	<i>fi</i>

La condition (booléenne) est en général le code de retour d'une commande UNIX.

Le code de retour de la commande détermine le test *if* :

- Code de retour valant zéro :
 Le test *if* est vrai.
- Code de retour non nul :
 Le test *if* est faux.

Exemple :

```
#!/bin/sh
if ls > /dev/null
then
    echo Il y a des fichiers
fi
```

C'est-à-dire sous sa forme générique :

```
#!/bin/sh
if commande [options] parametres
then
    ...
else
    ...
fi
```

Autres formes de *if* : les opérateurs logiques du shell

ET logique :

```
command1 && command2

if command1
then
    command2
fi
```

OU logique :

```
command1 || command2

if ! command1
then
    command2
fi
```

Avantage : forme compacte et élégante

Exemples :

1. `#!/bin/sh`
`ls foo && cat foo`

2. `#!/bin/sh`
`ls foo 2> /dev/null || echo "foo n'existe pas"`

§7.10 Structure **case**

La commande `case` permet de tester une chaîne de caractères par rapport à un certain nombre d'autres chaînes prédéfinies :

```
case value in
  pattern1) commands1
            ;;
  pattern2) commands2
            ;;
esac
```

Exemple :

```
#!/bin/sh
echo -n "Quitter ? --> "
read answer
case "$answer" in
    Y* | y* ) ANSWER="YES" ;;
    N* | n* ) ANSWER="NO" ;;
    b?? ) ANSWER="BOF" ;;
    * ) ANSWER="MAYBE" ;;
esac
echo $ANSWER
```

§7.11 Commande test

Dans de nombreuses structures shell, on teste une condition.

La commande **test** permet de réaliser divers tests.

Format	Description
-f fichier	True if fichier exists and is a regular file.
-s fichier	True if fichier exists and has a size greater than zero.
-w fichier	True if fichier exists and has write flag on.
-x fichier	True if fichier exists and has execute flag on.

Format	Description
<code>-n string</code>	True if the length of string is nonzero.
<code>s1 = s2</code>	True if the strings s1 and s2 are identical.
<code>s1 != s2</code>	True if the strings s1 and s2 are not identical.
<code>n1 -eq n2</code>	True if the integers n1 and n2 are algebraically equal.
<code>n1 -ne n2</code>	True if the integers n1 and n2 are not algebraically equal.
<code>n1 -gt n2</code>	True if the integer n1 is algebraically greater than the integer n2.
<code>n1 -ge n2</code>	True if the integer n1 is algebraically greater than or equal to the integer n2.
<code>n1 -lt n2</code>	True if the integer n1 is algebraically less than the integer n2.
<code>n1 -le n2</code>	True if the integer n1 is algebraically less than or equal to the integer n2.

Format	Description
<code>! expression</code>	True if expression is false.
<code>expression1 -a expression2</code>	True if both expression1 and expression2 are true.
<code>expression1 -o expression2</code>	True if either expression1 or expression2 are true.

Double forme de la commande test :

1. forme normale :

```
#!/bin/sh
if test $1 = hello
then
    echo hello world
fi
```

2. forme crochet :

```
#!/bin/sh
if [ $1 = hello ]
then
    echo hello world
fi
```

```
% ls -li /bin/[ /bin/test
```

```
15422 -r-xr-xr-x  2 bin  bin  45056 May 20 12:31 /bin/*
```

```
15422 -r-xr-xr-x  2 bin  bin  45056 May 20 12:31 /bin/test*
```

Par contre pas de commande Unix "]", bien sûr.

Exemples :

1. #!/bin/sh

```
if test $1 = hello
then
    echo hello world
fi
```

2. if [\$1 -gt \$2 -o \$1 -eq \$2]

```
then
    echo $1 is greater than or equal to $2
fi
```

3. #!/bin/sh

```
[ $# -eq 0 ] && echo You entered no parameters
```

§ 7.12 Structure de boucles : *while*, *for*, *until*

Syntaxes :

<pre>while condition do commands1 done</pre>	<pre>for variable in list do commands1 done</pre>	<pre>until condition do commands1 done</pre>
--------------------------------------------------	-------------------------------------------------------	--------------------------------------------------

Exemples :

<pre>1a) #!/bin/sh while ["\$1"] do echo \$1 shift done</pre>	<pre>% ./foo a b "two words" d e a b two words d e</pre>
<pre>1b) #!/bin/sh count=5 while [\$count -ge 0] do echo \$count count=`expr \$count - 1` done</pre>	<pre>% ./foo 5 4 3 2 1 0</pre>

2a) #!/bin/sh	% ./foo a "two words" d
echo \$#	3
for VAR in \$*	a
do	two
echo \$VAR	words
done	d
2b) #!/bin/sh	% ./foo a b "two words" d e
count=0	argv[1]=a
for i in "\$@"	argv[2]=b
do	argv[3]=two words
count=`expr \$count + 1`	argv[4]=d
echo "argv[\$count]=\$i"	argv[5]=e
done	

3a) #!/bin/sh	% ./foo a b "two words" d e
until ["\$1" = ""]	a
do	b
echo \$1	two words
shift	d
done	e
3b) #!/bin/sh	% ./foo
count=5	5
until [\$count -lt 0]	4
do	3
echo \$count	2
count=`expr \$count - 1`	1
done	0

§ 7.13 Contrôle du flux d'exécution : *break, continue*

Deux commandes du shell permettent de contrôler le flux d'exécution du code :

– **break** ou **break n**

Cela permet de sortir prématurément d'un niveau de boucle ou de $\langle n \rangle$ niveaux de boucles.

```
#!/bin/sh
count=5
while :
do
    [ $count -lt 0 ] && break
    echo $count
    count=`expr $count - 1`
done
```

On notera l'expression **while :**. Le caractère ":" correspond à une instruction builtin du shell dont le code de retour est toujours 0.

– **continue** ou **continue n**

Cela permet de retourner à la boucle sans finir le code de la boucle.

```
#!/bin/sh
for i in "$@"
do
    [ -d $i ] && continue
    echo "$i n'est pas un directory"
done
```

§ 7.14 Redirection dans les boucles

On peut réaliser des redirections au niveau des commandes `if`, `while`, `until` du shell et pas uniquement au niveau des commandes Unix.

```
1) #!/bin/sh
   if true
   then
       read line1
       read line2
       read line3
   fi < $1
```

```
2a) #!/bin/sh
    for file in "$@"
    do
        while read word1 rest_of_line
        do
            [ "$word1" = "Subject:" ] && \
                { echo "$file $word1 $rest_of_line" ; break; }
        done < $file
    done
```

```
2b) #!/bin/sh
count=0
while read BUFFER
do
    count=`expr $count + 1`
    echo "$count $BUFFER"
done < $1
```

```
3) #!/bin/sh
until [ "$word1" = ABORT ]
do
    read word1 rest
done < $1
```

§ 7.15 Fonctions shell

On peut écrire des fonctions au sein d'un shell script (sauf avec `/bin/sh` d'Ulrix, lui préférer `/bin/sh5`) :

```
#!/bin/sh

fonction()
{
    commands
}
```

Particularités :

- `$1`, `$2` ... désignent dans une fonction les paramètres passés à la fonction et non pas au shell script
- `$0` désigne toujours le nom du shell script qui tourne
- `$*` et `$@` désignent les listes des paramètres passés à la fonction, avec toujours les mêmes subtilités entre les deux.
- Il n'y a qu'un environnement de variables au sein d'un shell script. Une variable créée au sein d'une fonction peut donc être accédée en dehors du corps de cette fonction. Il n'y a pas de notion de variable locale en général.

<pre>#!/bin/sh foobar() { a=33 ; echo "b=\$b" ; } echo "a=\$a" b=2 foobar echo "a=\$a"</pre>	<pre>% ./foo a= b=2 a=33</pre>
--------------------------------------------------------------------------------------------------	--------------------------------

Code de retour d'une fonction :

La commande **return** renvoie une valeur de retour pour la fonction shell.

```
#!/bin/sh

foobar()
{
    return 1
}

if foobar
then
    echo Test positif
else
    echo Test negatif
fi
```

Le code de retour d'une fonction shell suit la même convention que pour les commandes Unix :

code de retour nul
la fonction s'est exécutée
correctement

code de retour non nul
la fonction a rencontré une condition
logique d'erreur

§ 7.16 Code de retour d'un shell script : **exit**

La commande **exit** renvoie une valeur de retour pour le shell script.

Le code de retour d'un shell script suit la même convention que pour les commandes Unix :

- code de retour nul
le script s'est exécuté correctement
- code de retour non nul
le script a rencontré une condition logique d'erreur

```
% cat /usr/bin/true
#!/bin/sh
#      @(#)true.sh 1.5 88/02/07 SMI; from UCB
exit 0
```

§ 7.17 Traitement des signaux : **trap**

Tout programme Unix a un comportement par défaut vis à vis des signaux qu'on peut lui envoyer.

La commande **trap** permet à un shell script de modifier son comportement par défaut vis à vis des signaux.

Syntaxe :

```
trap
trap commands signals
```

Exemples :

<pre>1. #!/bin/sh trap "echo SIGINT trapped ;" 2 trap sleep 100 % ./foo</pre>	<pre>2: echo SIGINT trapped ; ^C SIGINT trapped %</pre>
<pre>2. #!/bin/sh trap "echo Bye bye... ;" 0 trap % ./foo</pre>	<pre>0: echo Bye bye... ; Bye bye... %</pre>

```

3. #!/bin/sh
   trap "echo Bye bye... ;" 0
   trap "echo SIGINT trapped ;" 2
   trap
   sleep 100
   echo On continue...

% ./foo
0: echo Bye bye... ;
2: echo SIGINT trapped ;
^C
SIGINT trapped
On continue...
Bye bye...
%
```

```

4. % cat foo
   #!/bin/sh
   LOG=/tmp/foo.$$
   trap "rm -f ${LOG} ;" 0
   cp /etc/passwd ${LOG}
   ls -l ${LOG}

% ./foo
-rw-r--r--  1 besancon  bin   38 Sep 14 23:03 /tmp/foo.4941
% ls -l /tmp/foo.4941
ls: /tmp/foo.4941: No such file or directory
```

NB : on notera l'emploi de \$\$ pour récupérer le PID du script en train de s'exécuter

§ 7.18 Debugging d'un shell script : `set -x`

2 méthodes :

1. "The most effective debugging tool is still careful thought, coupled with judiciously placed print statements." (Brian Kernighan [1978])
2. Placer en début de script la ligne

```
set -x
```

Exemple :

<pre>% cat foo #!/bin/sh set -x echo \$USER</pre>	<pre>% ./foo + echo besancon besancon</pre>
---------------------------------------------------	---------------------------------------------

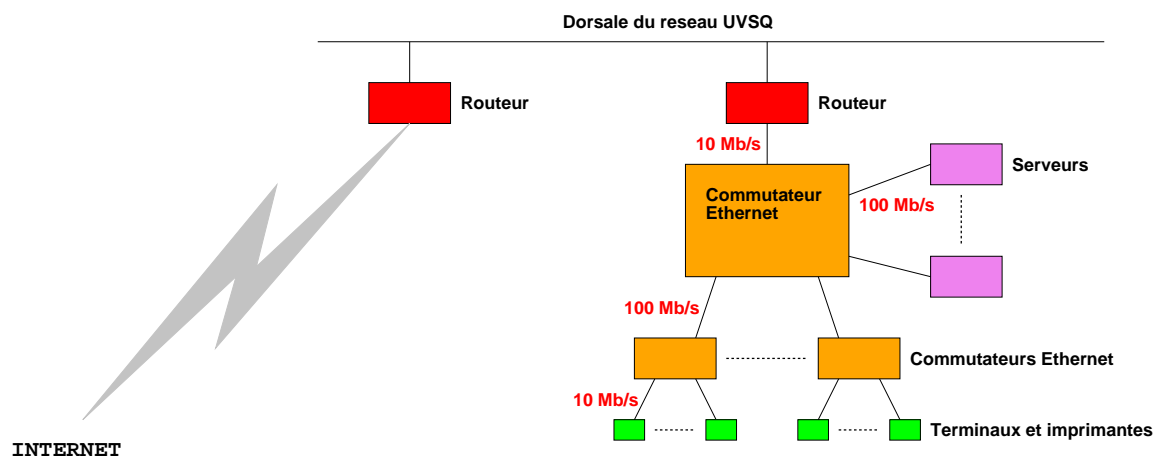
On peut aussi utiliser "`set -v`".

Pour le fun, deviner ce que fait ce shell script :

```
#!/bin/sh -- # set i=echo;set I='u[Cu[Cu[C';set l="tr u \033";$L      .-.
clear;cat $0;cat $0|sed '/D/d;s/L.*$/l;/s/.*# //;s/l/i;71H/g'|csh -f;[   V   ]
# while 2;$i "u[31/$I\u[21 $I " |$l;$i "u[31 $I u[21_${I}_"|$L      (( ))
# end;$i "u[31 $I u[21\${I}/"|$l;$i "u[21_${I}_"|$L  -yes@ludd.luth.se-  ^ ^
```

Chapitre 8 : Travail en réseau à l'UVSQ et sur Internet

§ 8.1 Raccordement Internet de l'UVSQ (plan de 1999)



Le Mb/s, c'est l'unité de débit autorisé par la technologie des cablages utilisés.

1 Mb/s : environ 100000 caractères circulant sur le câble en une seconde

Le réseau local est toujours plus rapide que le réseau vous raccordant à Internet :

- Le réseau d'enseignement mélange des tronçons à 10 Mb/s et à 100 Mb/s.
- L'UVSQ est reliée à Internet par une liaison à 2 Mb/s (200000 caractères par secondes maximum).

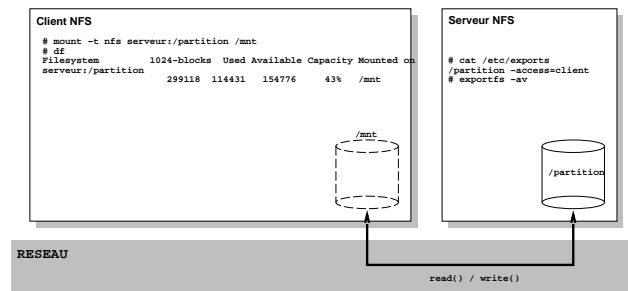
Merci de ne pas en abuser car cette liaison ne sert pas qu'aux étudiants mais aussi à tous les laboratoires de recherche de l'UVSQ et à l'administration.

Bien sûr :

- Le réseau est réservé à une utilisation professionnelle.
- Les communications peuvent être tracées à des fins de sécurité
- On ne se connecte pas à une machine sans autorisation

§ 8.2 Accès à vos fichiers, NFS, *df*, *du*, *quota*

Vos fichiers sont stockés sur un serveur dit *serveur NFS* (NFS \equiv Network Filesystem Share). Ils sont stockés en un unique point et rendus accessibles à tous les autres ordinateurs via un mécanisme système qui ne sera pas abordé ici.



Vous accédez de façon transparente à vos fichiers quelle que soit la machine sur laquelle vous vous connectez.

◇ Remplissage du disque

La commande *df* vous indiquera le taux de remplissage des disques durs locaux de l'ordinateur ainsi que des disques durs réseau qui stockent vos fichiers.

Par exemple (exemple pris sur une machine qui ne fait pas partie du réseau de l'UVSQ) :

```
% df
Filesystem                1k-blocks      Used Available Use% Mounted on
/dev/dsk/c0t0d0s0           123231        40131      70777   36% /
/dev/dsk/c0t0d0s3          1523574       702902     759730   48% /usr
/dev/dsk/c0t0d0s4           492422       392745     50435   89% /var
lucifer.prism.uvsq.fr:/src
                           2102787    1999249      96529   95% /src
lucifer.prism.uvsq.fr:/USERS/prism/staff
                           4375366    3006292    1325321   69% /users/prism/staff
torquenada.prism.uvsq.fr:/OPT/tmp
                           3918337    3282432    244075   93% /mnt/qic
```

◇ Calcul de la place disque occupée

Vous ne devez pas laisser votre compte se remplir de fichiers. Les disques durs n'ont pas une capacité infinie et hors de question de stocker toute la documentation disponible sur Internet chez vous !

La commande «`du -k $HOME`» vous donnera la taille disque que votre homedirectory occupe. La commande passe en revue tous les répertoires et en affiche la taille.

Le résultat affiché est exprimé en kilo octets (1 ko = 1024 octets).

```
% du
372      ./csi
107      ./cru
1793
```

◇ Quota

Comme les disques durs n'ont pas une capacité infinie et que des abus sont régulièrement constatés malgré les consignes données, un système de quota est activé.

Vous avez droit au maximum à stocker 35 Mo sur les ordinateurs de l'UVSQ.

La commande `quota -v` vous donnera des indications sur l'utilisation du quota que l'on vous a accordé.

En cas de problème avec votre quota, contactez les interlocuteurs du CSI (cf chapitre 1).

1. En dessous de la limite tolérée du quota :

```
% quota -v besancon
Disk quotas for besancon (uid 4332):
Filesystem      usage  quota  limit  timeleft  files  quota  limit  timeleft
/sae2           24903  30000  35000                11480  20000  25000
```

2. Vous dépassez la limite inférieure tolérée mais pas la limite supérieure du quota :

```
% quota -v besancon
Disk quotas for besancon (uid 4332):
Filesystem      usage  quota  limit  timeleft  files  quota  limit  timeleft
/sae2           31893  30000  35000  7.0 days  11520  20000  25000
```

Vous avez 7 jours pour faire du ménage pour retomber en dessous de la limite inférieure.

3. Vous atteignez la limite supérieure du quota :

```
% operation-consommatrice-de-place-disque
```

```
DISK LIMIT REACHED (/sae2) for uid 4332 pid 18742 - WRITE FAILED
cp: gros-fichier: Disc quota exceeded
```

Pas de cadeau : vous ne pouvez plus grappiller de place disque. Vous êtes bloqué dès maintenant.

```
% quota -v
Disk quotas for besancon (uid 4332):
Filesystem      usage  quota  limit  timeleft  files  quota  limit  timeleft
/sae2           34999  30000  35000  7.0 days  11540  20000  25000
```

4. Si vous n'avez pas fait le ménage au bout de 7 jours, que cela soit dans le cas 2 ou le cas 3 :

```
% quota -v besancon
Disk quotas for besancon (uid 4332):
Filesystem      usage  quota  limit  timeleft  files  quota  limit  timeleft
/sae2           34999  30000  35000  EXPIRED  11540  20000  25000
```

Plus de cadeau : vous êtes bloqué et devez faire du ménage.

◇ /public

Vous disposez de la zone /public qui vous permet de stocker des choses encombrantes qui surchargeraient votre homedirectory et feraient éclater vos quotas.

Attention : cette zone /public n'est pas sauvegardée contrairement à votre homedirectory. A vos risques et périls de l'utiliser.

§ 8.3 Impression : lpr, lpq, lprm

Une impression nécessite de connaître le nom de l'imprimante et d'avoir un fichier au bon format à imprimer.

Pour imprimer, utiliser la commande `lpr -Pimprimante fichier`

Pour consulter la queue d'impression, utiliser la commande `lpq -Pimprimante`

Pour retirer un fichier de la queue d'impression, utiliser la commande `lprm -Pimprimante numéro-dans-la-queue-renvoyé-par-lpq`

En cas de problème matériel avec une imprimante, contactez les interlocuteurs du CSI (cf chapitre 1).

§ 8.4 Nom de machine : `uname`, `hostname`

Une machine Unix a un nom qui permet de la désigner simplement dans des commandes de connexion réseau.

Pour connaître le nom de votre machine, vous pouvez employer les commandes :

- `hostname` :

```
% hostname
choc.unix.fr
```

- `uname -n` ; cette commande renvoie aussi le nom de la version d'Unix utilisé :

```
lancer uname -a
```

```
% uname -n
choc.unix.fr
```

```
% uname -a
```

```
SunOS choc.unix.fr 4.1.4 7 sun4m unknown
```

<http://www.csi.uvsq.fr/services/pedagogie/faq-s.html>

pour plus de renseignements sur les noms de machines des salles de licence.

§ 8.5 Tests de connectivité : `traceroute`, `ping`

Les machines distantes ne sont pas toujours disponibles :

- Elles peuvent être en panne.
- Le réseau qui permet de les joindre peut être en panne.
- etc.

Quelques commandes permettent de tester si une machine est joignable :

- Commande `ping machine`.

```
% ping vangogh
sendto: Network is unreachable
% ping choc
choc.unix.fr is alive
```

– Commande *traceroute* machine

Elle renvoie les intermédiaires réseau qui route notre acheminement vers la machine distante :

```
% traceroute jungle.ens.uvsq.fr
traceroute to jungle.ens.uvsq.fr (193.51.26.5), 30 hops max, 40 byte pack
 1  yacht (129.199.96.254)  0 ms  0 ms  0 ms
 2  renater (129.199.1.10)  2 ms  2 ms  2 ms
 3  195.83.239.157 (195.83.239.157)  8 ms  3 ms  3 ms
 4  danton1.rerif.ft.net (193.48.58.49)  14 ms  8 ms  8 ms
 5  danton2.rerif.ft.net (193.48.58.5)  8 ms  9 ms  200 ms
 6  stamand2.rerif.ft.net (193.48.75.5)  139 ms  10 ms  11 ms
 7  193.48.53.186 (193.48.53.186)  10 ms  10 ms  11 ms
 8  193.48.53.178 (193.48.53.178)  13 ms  17 ms  13 ms
 9  195.83.240.222 (195.83.240.222)  14 ms  87 ms  26 ms
10  r-ens.reseau.uvsq.fr (193.51.24.3)  134 ms  41 ms  28 ms
11  jungle.ens.uvsq.fr (193.51.26.5)  87 ms  30 ms  31 ms
```

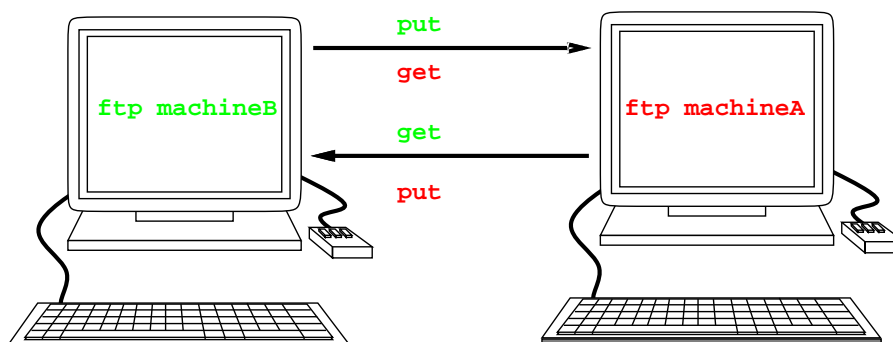
Le nombre d'intermédiaires n'est pas proportionnel à l'éloignement géographique de la machine destination.

§ 8.6 Transfert de fichiers entre machines, *ftp*

On peut être amené à transférer des fichiers entre deux machines si celles-ci ne partagent pas les mêmes disques réseau.

La commande à utiliser est *ftp machine*.

FTP ≡ File Transfer Protocol



◇ Sous commandes de ftp

La commande `ftp` vous place dans une espèce de shell dans lequel vous disposez des commandes suivantes (ce sont les plus importantes à votre niveau) :

- commande `binary` : à utiliser si le fichier à transmettre contient des caractères non texte
- commande `dir` : pour lister les fichiers sur la machine distante
- commande `cd directory` : pour changer de répertoire sur la machine distante
- commande `lcd directory` : pour changer de répertoire sur la machine locale
- commande `get fichier` : pour récupérer sur la machine distante un fichier
- commande `put fichier` : pour déposer sur la machine distante un fichier
- commande `quit` : pour se déconnecter

```
% ftp choc
Connected to choc.unix.fr.
220 choc.unix.fr FTP server (SunOS 4.1) ready.
Name (choc:besancon):
331 Password required for besancon.
Password:
230 User besancon logged in.
ftp> dir
200 PORT command successful.
150 ASCII data connection for /bin/ls (192.168.0.1,3945) (0 bytes).
total 307
-rw-r--r--  1 besancon software      69 Mar 20  1995 .Xdefaults
...
ftp> get fichier [nouveau nom]
...
ftp> put fichier [nouveau nom]
200 PORT command successful.
150 ASCII data connection for fichier (192.168.0.1,3947).
226 ASCII Transfer complete.
local: fichier remote: fichier
802 bytes sent in 0.0042 seconds (1.9e+02 Kbytes/s)
ftp> quit
221 Goodbye.
```

§ 8.7 Connexion shell sur des machines distantes : *telnet*, *ssh*

Quelques commandes permettent d'ouvrir une session shell sur une machine distante.

- Commande `ssh machine`.

C'est une commande très pratique car elle chiffre la communication avec la machine distante évitant ainsi les piratages du type mouchard réseau qui récupère les mots de passe :

```
% ssh dmi
besancon@dmi's password:
Last login: Sat Sep 15 22:00:11 2001 from ppp-2
Sun Microsystems Inc. SunOS 5.5 Generic November 1995
No mail.
dmi%
```

- Commande `telnet machine`.

```
% telnet dmi
Trying 129.199.96.11...
Connected to dmi.ens.fr.
Escape character is '^]'.
```

SunOS 5.7

```
login: besancon
Password:
Last login: Sat Sep 15 21:54:15 from ppp-2
Sun Microsystems Inc. SunOS 5.5 Generic November 1995
dmi%
```

A chaque fois que ce sera possible, préférer une connexion shell distante en utilisant *ssh*.

§ 8.8 Liste d'utilisateurs connectés : *users, who, w*

Plusieurs commandes permettent d'avoir la liste des personnes connectées sur un système Unix :

- commande *users*
- commande *who*
- commande *w*

Les renseignements renvoyés ne sont pas de la même forme selon la commande utilisée.

§ 8.9 Courrier électronique, adresse, *mutt, netscape*

Votre adresse de courrier électronique est du type

Prenom.Nom@ens.uvsq.fr.

Pour lire son courrier, vous disposez des utilitaires :

- *mutt*
Affichage en mode texte de vos mails. Gère autant de son mieux les pièces attachées.
- *netscape*
Affichage graphique de vos mails. Plus consommateur de ressources. Attention, il fait «exploser» certains terminaux sous configurés. Ce genre de logiciels est la tendance du monde Windows...

§ 8.10 Web, URL, lynx, netscape

Comment consulter une documentation sur Internet ?

Les renseignements sont indiqués par un URL, Universal Resource Locator :

Dénomination unique à caractère universel qui permet de localiser une ressource, un document, sur l'Internet, et qui indique la méthode pour y accéder, le nom du serveur et le chemin à l'intérieur du serveur.

Typiquement :

`protocole://serveur/chemin`

Les types les plus répandus :

`http://www.lemonde.fr/`

`mailto:Thierry.Besancon@prism.uvsq.fr`

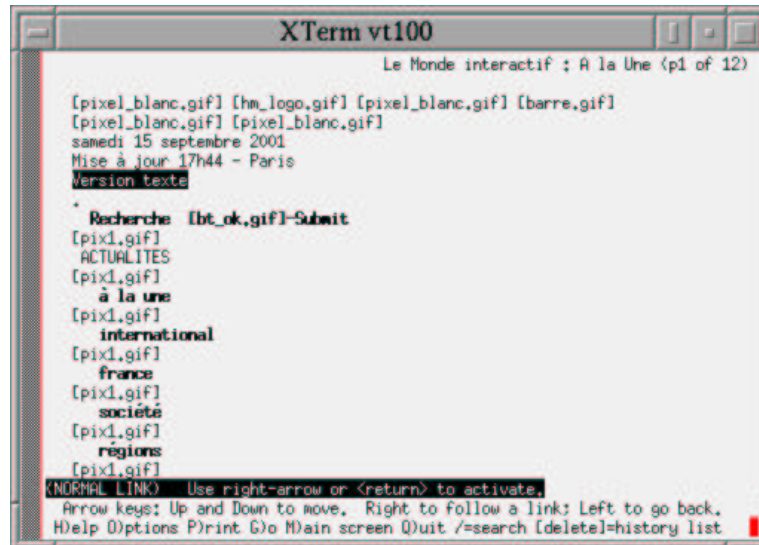
`ftp://ftp.uvsq.fr/`

Pour consulter un URL, on utilise un *browser* Web. Les plus connus :

- Netscape (sur Unix, Macintosh et Windows) ; fonctionne en mode graphique
- Internet Explorer (sur Macintosh et Windows) ; fonctionne en mode graphique
- lynx (sur Unix) ; fonctionne en mode texte

Attention, dans les salles d'enseignement de licence, la commande `netscape` fait souvent «exploser» certains terminaux sous configurés...

Il suffit alors d'entrer l'URL à consulter et on se retrouve connecté au serveur (si on peut le joindre)



Chapitre 9 : Système de multifenêtrage : X

Système de multifenêtrage (Window System) :

ensemble de programmes, de bibliothèques de programmation réalisant une interface homme / machine bâtie sur l'utilisation de divers équipements :

- clavier
- souris
- écran graphique
- autres périphériques (spaceball, plaquette graphique, ...)

L'écran tente de réaliser un modèle de bureau.

L'idée vient des travaux de Xerox, repris par Apple, repris par le projet Athena du MIT.

Pour Unix, le système de multi fenêtrage est un système appelé **X** ou X Window System ou X11. Ce n'est pas "X Windows" ! Greuh !!!

Sites de distribution gratuite :

- <ftp://ftp.x.org>
- <http://www.x.org>
- <ftp://ftp.uvsq.fr/pub/X11> (site miroir)

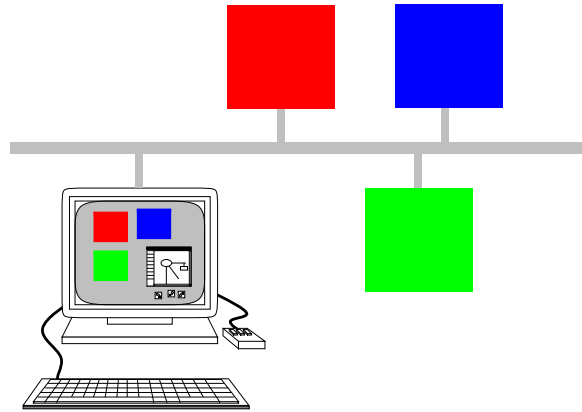
Newsgroups :

- `comp.windows.x`
- `comp.windows.x.announce`
- `comp.windows.x.apps`
- `comp.windows.x.i386unix`
- `comp.windows.x.intrinsics`
- `comp.windows.x.motif`
- `comp.windows.x.openlook`
- `comp.windows.x.pex`

§ 9.1 Caractéristiques de X

C'est un peu plus qu'un système de multifenêtrage : il est **réparti**.

L'interface homme / machine permet de travailler sur plusieurs systèmes à la fois :



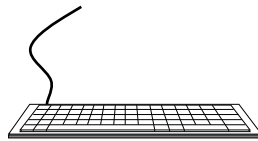
X est indépendant :

- des matériels
- des systèmes d'exploitation
- des protocoles de communication (TCP/IP mis aussi DecNet...)

X fournit des mécanismes.

X ne définit pas de méthodes.

§ 9.2 Clavier



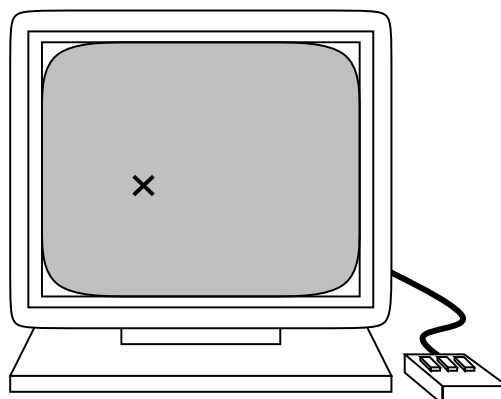
Plusieurs modificateurs :

- Control
- Shift
- Meta
- CapsLock
- ...

Tous les types de claviers sont supportés.

Au pire, on peut configurer un clavier hors du commun via la commande **"xmodmap"**.

§ 9.3 Souris

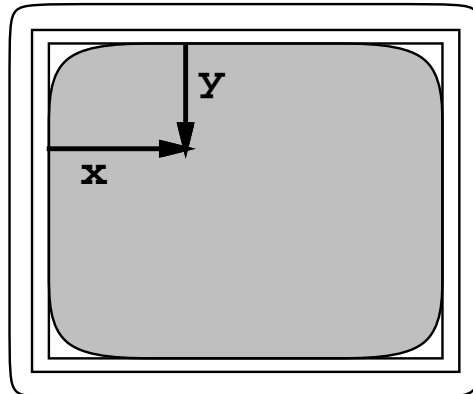


C'est un outil de désignation ayant de 1 à 3 boutons.

La position est suivie à l'écran par un curseur.

Le curseur est toujours affiché au premier plan.

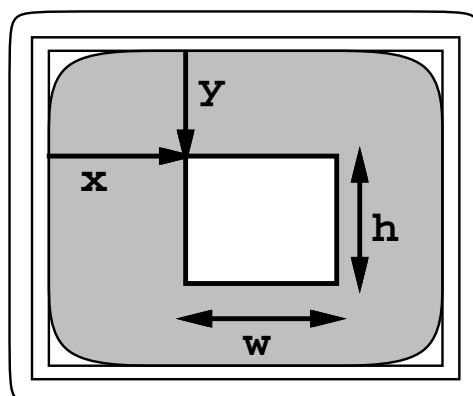
§ 9.4 Ecran



Il est de type **bitmap** (affichage point).

L'image est obtenue par balayage d'une mémoire d'écran (frame buffer) contenant une valeur par point à l'écran (pixel).

§ 9.5 Fenêtre



Rectangle à l'écran caractérisé par un emplacement, une bordure, ses dimensions.

§ 9.6 Icône

C'est une petite fenêtre.

C'est la trace visible à l'écran d'une fenêtre temporairement non affichée.

§ 9.7 DISPLAY

C'est un triplet (écran, clavier, souris)

C'est une station de travail Unix, un terminal X, un mac avec le bon logiciel (Mac X, eXodus), un PC avec le bon logiciel (cf <ftp://ftp.lip6.fr/pub9/doc/faqs/x-faq/Intel-Unix-X-faq.gz>)

Un display a un nom et une adresse.

L'adresse est une adresse IP.

Son nom est du type "linux7.formation.jussieu.fr:0.0".

0.0 désigne l'écran 0 de la machine. X permet de gérer des machines multicéphales.

§ 9.8 Architecture de X

X est constitué des parties suivantes :

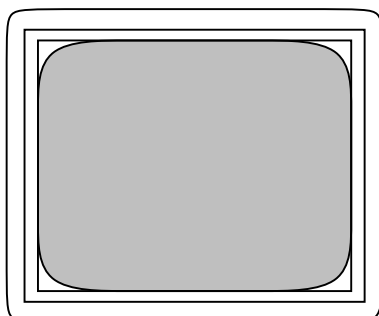
- un serveur
C'est un programme gérant du matériel (le display) et du pur logiciel : fenêtres, ressources, événements.
- des clients
Des applications utilisant un serveur X.
- un protocole
Il fait communiquer les clients et le serveur.
- des bibliothèques
Elles réalisent le protocole et l'interface des clients

§ 9.9 Serveur X

Le serveur X gère le display mais aussi :

- le système de communication sous jacent
- les particularités du système d'exploitation

Il gère un ensemble de fenêtres organisées en arbre dans lequel les propriétés se transmettent par filiation.



La racine de l'arbre est la **root window** créée à l'initialisation du serveur et couvrant tout l'écran.

§ 9.10 Clients X

Ce sont les programmes basés sur les bibliothèques X.

Le protocole X est utilisé pour envoyer des requêtes au serveur X qui réalisera leurs fonctions graphiques.

Ce sont les clients qui réalisent l'interface homme / machine de X

(contrairement à Windows, à Mac OS où l'interface est intégrée au système d'exploitation).

Quelques clients essentiels :

- émulation graphique d'un terminal :
xterm
- gestionnaire de fenêtres :
twm, tvtwm, fvwm, mwm, olwm, olvwm, ...
- accessoires de bureau :
xclock, xbiff, ...

§ 9.11 Gestionnaire de fenêtres

Gestionnaire de fenêtres \equiv Window Manager

C'est un client X, au même titre que les autres. Il y en a plein. Cf annexe E.

Il permet de réaliser les choses suivantes :

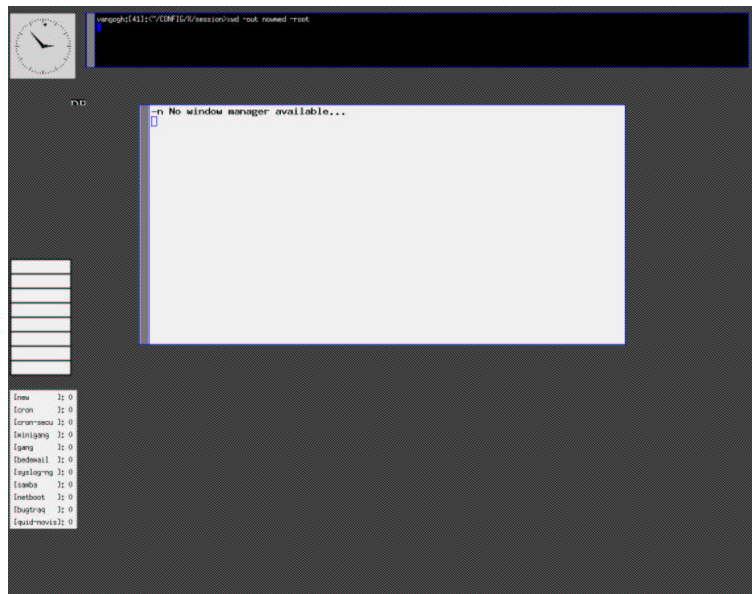
- déplacer ou redimensionner une fenêtre
- iconifier ou non une fenêtre
- faire passer au premier ou au dernier plan une fenêtre
- décorer les fenêtres
- créer ou détruire les fenêtres
- lancer ou terminer des applications

L'ICCM (InterClient Communication Manual) définit les règles de cohabitation entre les clients et le gestionnaire de fenêtres.

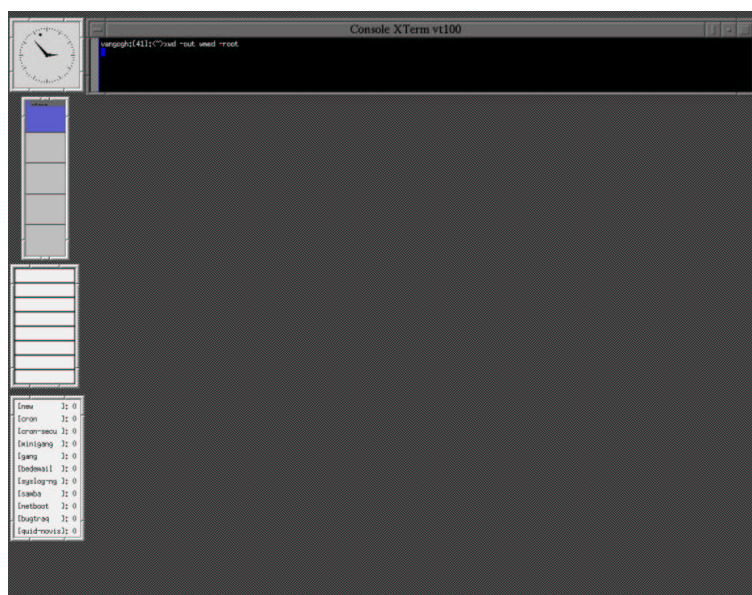
Il ne faut pas confondre le gestionnaire de fenêtres avec le serveur X.

L'emploi d'un gestionnaire de fenêtres n'a rien d'obligatoire mais ce serait se priver de beaucoup de fonctionnalités.

Ecran sans gestionnaire de fenêtres :



Ecran avec un gestionnaire de fenêtres :

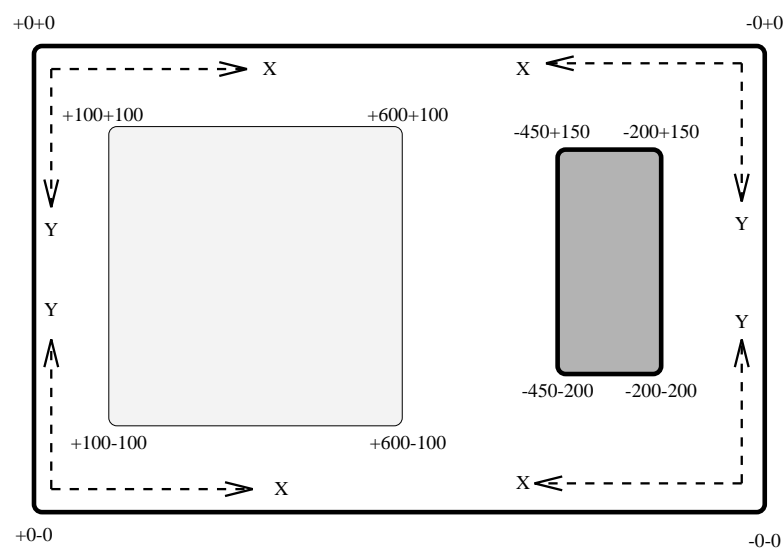


§ 9.12 Spécification des options aux clients X

Il y a des options standard pour la plupart des clients (tout au moins ceux reposant sur l'emploi de la bibliothèque Xt) :

- couleur des caractères :
"-fg couleur"
- couleur de fond des fenêtres :
"-bg couleur"
- caractères :
"-fn font"

- dimensions et emplacement de la fenêtre :
"-geometry WIDTH×HEIGHT±XOFF±YOFF"



§ 9.13 Spécification des couleurs aux clients X

On peut désigner une couleur sous un nom symbolique : cf

`/usr/X11R6/lib/X11/rgb.dir`

`/usr/X11R6/lib/X11/rgb.pag`

`/usr/X11R6/lib/X11/rgb.txt`

On peut aussi donner une couleur sous la forme de ses composantes Rouge Vert Bleu (RGB en anglais) :

`rgb: <red> / <green> / <blue>`

Ancienne syntaxe supportée pour compatibilité :

<code>#RGB</code>	(4 bits each)
<code>#RRGGBB</code>	(8 bits each)
<code>#RRRGGBBB</code>	(12 bits each)
<code>#RRRRGGGGBBBB</code>	(16 bits each)

Cf annexe F pour un exemple.

Principal problème avec les couleurs : le flash suite aux changements de colormap.

Raison : une colormap a un nombre limité de cellules.

Si une application a besoin de plus de cellules que disponibles, l'application alloue une colormap privée qui est installée à chaque fois que l'on se trouve dans une de ses fenêtres et desinstallée quand on les quitte. D'où un flash.

L'intérêt d'un écran 24 bits est de permettre de ne pas avoir ce flash de colormap. Son intérêt ne réside pas dans la disponibilité de nombreuses couleurs dont les nuances sont indiscernables de l'oeil humain.

§ 9.14 Spécification des polices de caractères

Les fontes ont des noms bien précis :

`-adobe-courier-medium-r-normal--10-100-75-75-m-60-iso8859-1`

(marque, nom, graisse, angle, largeur, style, taille en pixels, taille en 1/10 points, resolution X, resolution Y, approche, taille moyenne, ensemble)

On peut utiliser la commande **xlsfonts** pour avoir la liste des fontes utilisables sur son poste X.

Plusieurs formats sont possibles :

- BDF (Bitmap Distribution Format), genre de format source
- SNF (Server Natural Font), format compilé à partir d'un source BDF par `bdftosnf`
- PCF (Portable Compiled Font), format compilé à partir d'un source BDF par `bdftopcf`
- Type1 PostScript depuis X11R6, apparu avec X11R5
- True Type (<http://www.freetype.org/>)
- PEX
- Speedo

§ 9.15 Personnalisation – Ressources X

Tous les clients X sont personnalisables au moyen de ressources.X

Il y a 2 types de ressources :

Ressources standard

Héritées de la bibliothèque de programmation Xt Intrinsics très utilisée.

Spécification de fontes, géométrie, background, foreground, borderwidth, display, ...

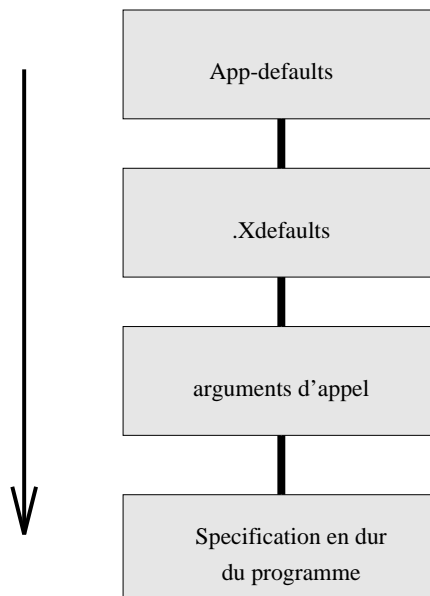
Ressources particulières à l'application

Par exemple, dans xterm, la possibilité d'afficher ou pas un ascenseur (ressource "scrollBar") ou de spécifier le nombre de lignes à garder en mémoire à l'affichage (ressource "saveLines").

Cf page de manuel de chaque application.

Où trouver les ressources ?

Les dernières ressources spécifiées
sont prioritaires



Les ressources par défaut pour les applications sont enregistrées dans des fichiers dans `/usr/X11R6/lib/X11/app-defaults/` par exemple sur Linux :

```
% ls /usr/X11R6/lib/X11/app-defaults/
Beforelight          XClock              XbmBrowser-color
Bitmap               XConsole            Xditview
Bitmap-color         XFontSel            Xditview-chrtr
Chooser              XLoad               Xedit
Clock-color          XLogo               Xfd
Editres              XLogo-color         Xgc
Editres-color        XPaint              Xmag
Viewres              XSm                 Xman
XCalc                XTerm               Xmessage
XCalc-color          XTerm-color         Xmh
XClipboard           XbmBrowser          Xvidtune
```

On notera les majuscules !

En ce qui concerne les ressources personnalisées par l'utilisateur, le fichier chez l'utilisateur les contenant est `$HOME/.Xresources`

On trouve aussi `$HOME/.Xdefaults`.

En pratique, faire un lien de `$HOME/.Xdefaults` vers `$HOME/.Xresources`.

La commande **xrdb** permet de manipuler les ressources chargées dans le serveur :

```
% xrdb -query > $HOME/ressources
% vi $HOME/ressources
% xrdb -load $HOME/ressources
```

A quoi ressemble une ressource ?

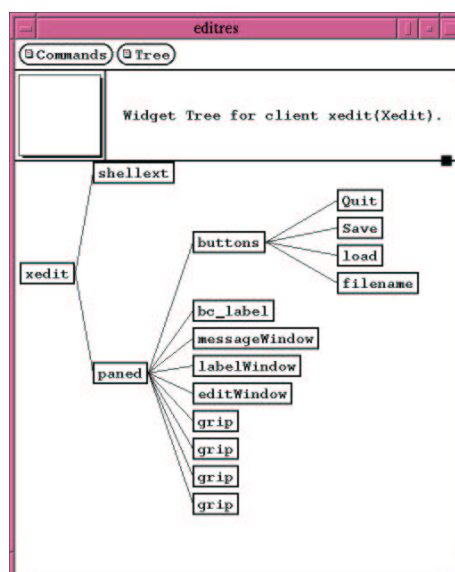
Exemple :

```
% xterm -fg red -sb -ls -bw 4 -fn 10x20
```

≡

```
xterm*scrollBar:      True
xterm*foreground:     red
xterm*loginShell:     True
xterm*borderWidth:    4
xterm*font:           10x20
```

Un client X est construit autour d'un arbre de widgets fournis par les bibliothèques :



Une ressource désigne donc, via une sorte de PATH, un élément dans l'arbre à personnaliser.

Les composantes du path de la ressource peuvent être séparées par '.' ou '*' :

```
Clock*Background:      grey
Clock*BorderColor:     light blue
Clock*hour:            yellow
Clock*jewel:           yellow
Clock*minute:          yellow
```

```
XTerm.JoinSession:     False
XTerm*mainMenu*quit*Label: Quit
```

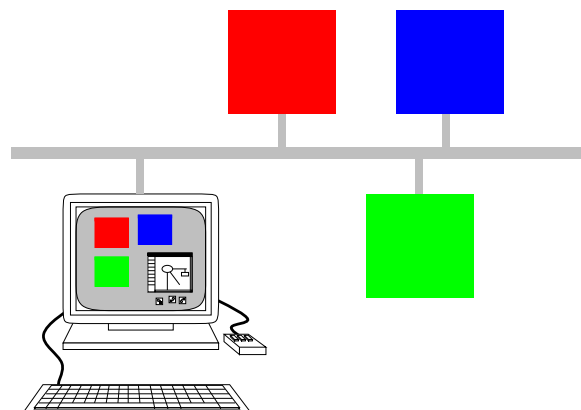
Le caractère '*' permet de sauter la spécification de composantes intermédiaires.

On peut aussi préciser des ressources via la ligne de commande :

```
% xterm -xrm 'XTerm*mainMenu*quit*Label: Quit'
```

§ 9.16 Autorisation d'accès

Principe client / serveur :



On doit pouvoir contrôler quels clients peuvent se connecter au serveur X.

La commande **xhost** permet de gérer les accès au serveur X :

```
% xhost
access control enabled, only authorized clients can connect
```

```
% xhost + peterpan.ens.fr
peterpan.ens.fr being added to access control list
```

```
% xhost
access control enabled, only authorized clients can connect
INET:peterpan.ens.fr
```

```
% xhost - peterpan.ens.fr
peterpan.ens.fr being removed from access control list
```

En pratique :

```
% echo $DISPLAY
```

```
MA-MACHINE:0.0
```

```
% xhost + FQDN
```

```
% rlogin FQDN
```

```
$ setenv DISPLAY MA-MACHINE:0.0
```

```
$ clientX
```

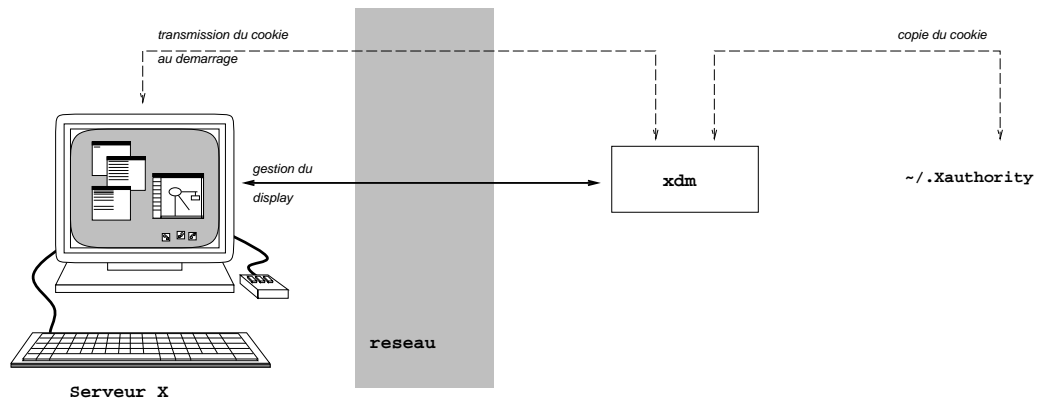
```
...
```

```
$ exit
```

```
% xhost - FQDN
```

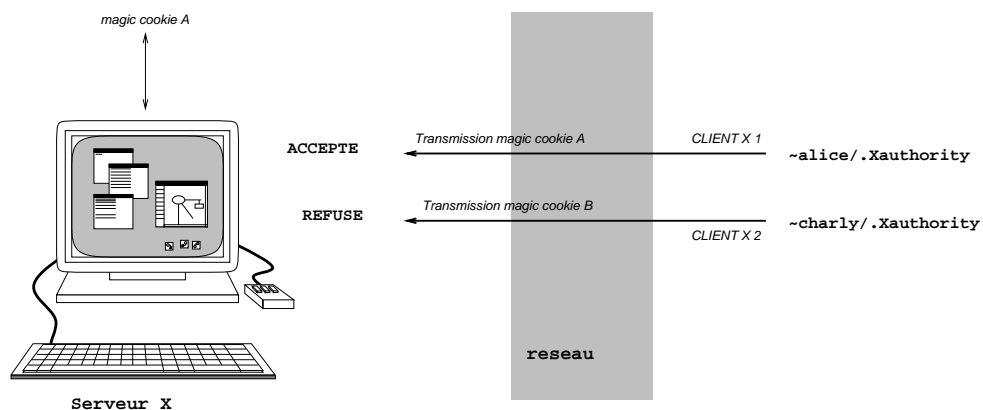
Le défaut de la commande xhost est d'autoriser une machine et donc tout utilisateur y étant connecté.

Solution : emploi d'un cookie, pseudo secret connu du serveur à rappeler lors d'une demande de connexion au serveur X.



Qui peut connaître le cookie ?

Qui peut accéder au contenu du fichier \$HOME/ .Xauthority!



⇒ La sécurité de X repose dans ce cas sur la sécurité du filesystem.

D'où :

```
% ls -l $HOME/.Xauthority
-rw----- 1 besancon software 1510 Dec 17 19:12 /users/adm/besancon/.Xauthority
```

Que faire si \$HOME/.Xauthority n'est pas disponible sur la machine distante ?

Réponse : utiliser la commande **xauth**

Extrait de la page de manuel :

The most common use for xauth is to extract the entry for the current display, copy it to another machine, and merge it into the user's authority file on the remote machine:

```
% xauth extract - $DISPLAY | rsh otherhost xauth merge -
```

Le danger : l'utilisateur faisant "xhost +"
⇒ tout client peut se connecter !

L'attaque :

- lecture / espionnage du clavier ⇒ récupération de mots de passe lors de "su root"
Cf
`ftp://ftp.giga.or.at/pub/hacker/unix/xscan.tar.gz`
- destruction de fenêtres par l'utilisation de "**xkill**" par exemple
- capture d'écran par l'utilisation de "**xwd -root**"

§9.17 Utilisation répartie de X

Manuellement

```
% echo $DISPLAY
```

```
MA-MACHINE:0.0
```

```
% xhost + FQDN
```

```
% rlogin FQDN
```

```
$ setenv DISPLAY MA-MACHINE:0.0
```

```
$ clientX
```

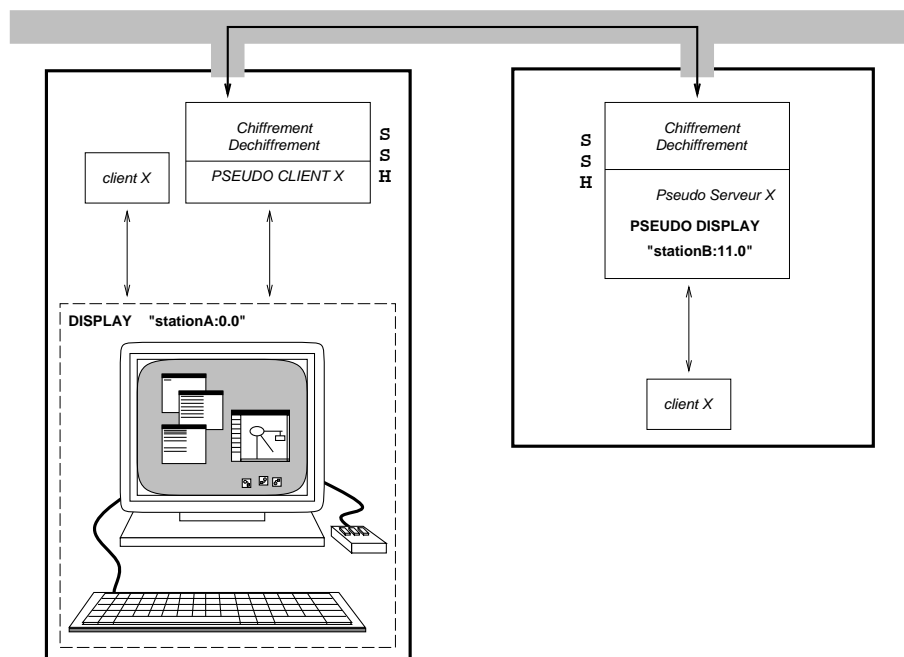
```
...
```

```
$ exit
```

```
% xhost - FQDN
```

On peut automatiser :

- xrsh : cf
ftp://ftp.lip6.fr/pub/X11/contrib/utilities/xrsh-5.8.shar.gz
- rxx : cf
ftp://ftp.lip6.fr/pub/X11/contrib/utilities/rxx-4.2.3.tar.gz
- SSF : transmission automatique du DISPLAY avec son cookie associé (voir <http://www.in2p3.fr/secur/ssf>) et sécurisation de la liaison par chiffrement légal de la communication :



```
% echo $DISPLAY
:0.0
% ssf central.pasteur.fr
    besancon@central.pasteur.fr's password:
    ...
% echo $DISPLAY
nefertiti.pasteur.fr:11.0
...
```

§ 9.18 XDM : X Display Management

C'est la fenêtre d'accueil standard sous X. Cf annexe G.

`xdm` \equiv `getty` + `login`

`xdm` est un programme qui tourne sur un serveur de calcul et qui gère un ensemble de serveurs X (par exemple sur des terminaux X).

`xdm` crée un shell sur le serveur de calcul qui configure l'environnement utilisateur via le fichier "`$HOME/.xsession`".

\Rightarrow **la durée de vie de la session sous X est celle du script**

`$HOME/.xsession`.

En général :

```
#!/bin/sh
appliX1 &
appliX2 &
...
appliXn &
appliX
```

Le dernier client X n'est pas lancé en tâche de fond. En général, le dernier client est un window manager.

Cf <ftp://ftp.lip6.fr/pub/doc/faqs/x-faq/speedups.gz>

S'il y a des erreurs, les messages de celles-ci sont écrits dans le fichier
\$HOME/.xsession-errors.

En cas d'erreur grave dans \$HOME/.xsession empêchant le démarrage de la session X, utiliser le mode **FailSafe** :

1. entrer le nom de login
2. valider par la touche Retour
3. entrer le mot de passe
4. valider par la touche **F1** et non pas par la touche Retour

Il apparaît alors un simple xterm sans window manager.