UNIX Résumé

Version 4.7 – Septembre 2007

Observatoire de Strasbourg

Marc Wenger

Préambule. septembre 1997

Avant de prendre ce document pour ce qu'il n'est pas et de lui trouver des tas de défauts, sachez que:

- Ce n'est pas un livre sur UNIX, mais un simple support de cours, qui a aussi un peu la prétention de pouvoir servir d'aide mémoire pour tout utilisateur d'UNIX. Vous n'y trouverez donc pas toutes les options de la commande `ls`. Ni des autres commandes d'ailleurs.
- L'écriture de ce document a été grandement facilitée par les notes de cours de 1992 de Philippe Paillou, ainsi que par un "UNIX primer", support de cours de *l'Université de Californie, Irvine*, trouvé sur un compte *ftp anonyme* et écrit par une personne tout aussi anonyme...

Version	1.0	sept	1993
	1.1	dec	1993
	2.0	oct	1995
	3.0	sept	1997
	3.1	sept	1998
	4.0	sept	1999
	4.1	sept	2000
	4.2	sept	2001
	4.3	sept	2002
	4.4	sept	2003
	4.5	sept	2004
	4.6	sept	2006
	4.7	sept	2007

Marc Wenger
Observatoire de Strasbourg
11, rue de l'université
F – 67000 STRASBOURG
e-mail: wenger@astro.u-strasbg.fr

*Contents

■ L'informatique. Généralités

le matériel

- Exécution des programmes: **Unité centrale - processeur**. CPU = Central Process Unit.
  Les processeurs sont caractérisés par la taille des mots adressés (8, 16, 32 ou 64 bits), leur vitesse d'horloge (GHz), et leur architecture et jeu d'instructions.
  Deux grandes familles:
  - ▷ **CISC** (Complex Instruction Set Computer): Pentium (Intel), Motorola 680x0, VAX
  - ▷ **RISC** (Reduced Instruction Set Computer): Sparc (SUN), Alpha (DEC), R6000 (IBM), PowerPC (IBM)

  Architecture:
  - ▷ **scalaire** : l'ensemble d'un programme est exécuté sur un processeur. Celui-ci peut avoir plusieurs unités arithmétique et logique pour exécuter simultanément plusieurs instructions
  - ▷ **vectoriel** : des opérations semblables sur les éléments d'un vecteur seront exécutées en parallèle
  - ▷ **parallèle** : des parties d'une application, définies explicitement ou non, sont exécutées sur des processeurs différents

  Mesure de la vitesse:
  - ▷ **MIPS**: Millions of Instructions per Second
  - ▷ **MFLOPS**, **GigaFlops**, **TeraFlops**: Millions of Floating Point instructions/sec (respectivement milliards et 10^{12} instructions flottantes/sec.)
  - ▷ **SpecMarks**: Mesure de puissance obtenus par un programme normalisé, intégrant plusieurs paramètres (cpu, entrées/sorties,...). **Specint2000**, **Specfp2000**.
  - ▷ **Linpack 100x100** et **1000x1000** (J. Dongarra): Programme standardisé de calcul matriciel mesurant la vitesse de calcul flottant d'un ordinateur. En tenant compte, le cas échéant, de ses capacités en calcul parallèle.
- Stockage des données et des programmes: **Mémoire centrale, mémoire vive**
  Une mémoire vive est caractérisée par sa taille, exprimée en Megaoctets (Mo) ou Gigaoctets (Go) et par sa vitesse d'accès (en nanosecondes).
- Organes d'interface avec le monde extérieur: **Périphériques**
  - ▷ **clavier – écran – souris**
  - ▷ **mémoires de masses à accès direct**:
    - ⋆ **disquettes**: faible capacité, micro-informatique. Divers autres systèmes de capacité 100 Mo à 2 Go.
    - ⋆ **clés USB**: capacité jusqu'à 8 Go.
    - ⋆ **disque dur**: 80 Go à 750 Go.
    - ⋆ **disque optique**:
      - · **CD-ROM**: écriture unique (650 - 700 Mo)
      - · **CD-RW**: écritures multiples (650 - 700 Mo)
      - · **DVD**: Digital Versatile/Video Disk (4.7 Go à 8.9 (19 ?) Go)

Utilisation possible de juke-box de quelques dizaines ou centaines de disques.

- ▷ **mémoires de masses à accès séquentiel**:
  - ⋆ **LTO** sauvegarde et stockage de haute capacité, 100 – 400 Go
  - ⋆ **DLT/SDLT** (Standard DEC/Quantum): sauvegardes et stockage. cartouches de 800 Go maximum. Juke-Box.
  - ⋆ **DAT** (Cassettes 4mm): sauvegardes, 12 – 36 Go
  - ⋆ **EXABYTE** (Cassettes vidéo 8mm): sauvegardes jusqu'à 80 Go
  - ⋆ **bande optique** : archives, 1 To (CREO) (1 Tera-octets $= 10^{12}$ octets). Très peu répandu.
- ▷ **imprimantes**:
  - ⋆ types: laser, à jet d'encre, à sublimation, matricielle à aiguilles
  - ⋆ langages: PDF, Postscript, HPPCL, HPGL, Canon, Tektronix, Sixel(DEC)
- ▷ **scanner**: saisie d'images, reconnaissance de texte
- ▷ **tablette graphique**: dessin manuel, saisie de tracés

le logiciel

Les langages:

- interprétés:
  - ▷ Langages de script shell UNIX: **sh**, **csh**, **bash**, **ksh**, ...
  - ▷ Langages de scripts à usage général: **Perl**, **Python**, **Ruby**, **Tcl/Tk**, **Javascript** ...
- compilés:
  - ▷ Calcul scientifique: **Fortran**
  - ▷ Universel, normalisé ANSI, toutes applications: **C**.
  - ▷ Extension orientée objets du langage C: **C++**
  - ▷ Orienté objet, indépendant plateforme, machine virtuelle : **Java**
  - ▷ Orienté objet, lié à Windows, machine virtuelle : **C#**
  - ▷ Universel, mal standardisé, académique: **Pascal**
  - ▷ Universel, peu utilisé, complexe, descendant de Pascal : **Ada**
  - ▷ Intelligence artificielle: **Lisp**, **Prolog**
  - ▷ Applications de gestion: **Cobol**
  - ▷ Langages anciens: **Algol**, **PL/1**
- spéciaux:
  - ▷ Langages machines: assembleurs
  - ▷ Langages de PAO: **Postscript**,**TEX**/**LATEX**, **HTML**, **nroff**/**troff**
  - ▷ Langages de gestion de documents web : **HTML**
  - ▷ Langages de description et d'échange de données : **XML**, **SGML**
  - ▷ Langage de base de données : **SQL**
  - ▷ Langages marginaux, mais historiquement importants: **Forth**, **Smaltalk**

Les systèmes d'exploitations:

- Non propriétaire, station de travail tous constructeurs : **UNIX**
- Open source, surtout PC : **Linux**, **FreeBSD**
- Propriétaire (Microsoft), PC: **MS-Dos**, **Windows 3.1/9x/NT/2000/XP/Vista**
- Propriétaire (Apple), MacIntosh: **Mac OS X**

Les réseaux

- Réseau commuté, lent: **réseau téléphonique** – 56 Kb/s
- Réseau commuté, multi-usage: **NUMERIS** – 2x64Ko/s
- Réseau commuté, rapide: **ADSL** – 512Kb/s – 20Mb/s
- Réseau à haut débit: **ATM** – 155 Mb/s
- Réseau local rapide : ETHERNET – 10 / 100 / 1000 Mb/s
- Réseau local Token Ring (IBM)
- Réseau local haute vitesse à fibre optique : FDDI
- Lignes spécialisées: tous débits. câble, fibre optique.
- Réseau académique à l'origine, transmission de données: **INTERNET**, protocole TCP/IP.

Les fonctions principales d'un système d'exploitation

- Programme de fond tournant en permanence sur la machine
- Gère les ressources de l'ordinateur:
  - ▷ **la cpu**:
    - ⋆ exécution des process (des programmes).
      - · multi-tâches = plusieurs process exécutables simultanément
      - · multi-utilisateurs = plusieurs utilisateurs connectables simultanément
      - · multi-processeur = plusieurs processeurs
    - ⋆ Ordonnancement des process. Quanta de temps. priorités.
    - ⋆ Indépendance des process. protection de tous contre le plantage d'un seul.
    - ⋆ Gestion des process.
      Etats: exécutable, stoppé, en attente de ressources (disque ou page mémoire), inactif, zombie.
    - ⋆ Comptabilité des process (*accounting*).
  - ▷ **la mémoire centrale**:
    - ⋆ Gestion de la mémoire centrale et de la mémoire virtuelle
    - ⋆ Allocation de mémoire à un process
  - ▷ **les disques**: Organisation et gestion des partitions et des fichiers.
  - ▷ **les périphériques**: Gérés par l'intermédiaire de drivers.
- L'initialisation d'un ordinateur est faite au moyen d'un programme codé dans l'unité centrale:
  le bootstrap

L'historique d'UNIX

- Ecrit initialement par Ken Thompson et Dennis Ritchie aux Bell Labs (AT&T) en 1969. Nouveau système inspiré de MULTICS (crée au MIT) pour mini-ordinateurs PDP.
- Le langage C a été développé en 1971 par Dennis Ritchie dans le but d'écrire UNIX de façon portable.
- UNIX a été reécrit en C en 1973
- UNIX édition 7 disponible en 1978 aux Bell Labs.
- Puis, divergence en deux familles:
  - ▷ Une version commerciale distribuée par AT&T à partir de 1983 appelée **System V**.
  - ▷ Une **versions BSD** développée par l'université de Berkeley sous license AT&T:
    - ⋆ BSD 4.1 (1981): ajout de `vi`, `csh` et la gestion de mémoire virtuelle
    - ⋆ BSD 4.2 (1983): amélioration des fonctions réseaux et communications entre process (*sockets*)
    - ⋆ BSD 4.3 (1987): ajout de la gestion de *jobs* en tâches de fond
- Le nom UNIX est propriété des Bell Labs.
  Des versions reécrites par d'autres vendeurs ont des noms différents, mais se rapportent à l'une des deux familles (System V ou BSD):

System V :	HP-UX	Hewlett-Packard	**BSD :**	Ultrix	Digital (DEC)
	AIX	IBM		SunOs 4.1.x	SUN
	Solaris 2.x	SUN			
	Dec Unix	DEC			

- Il existe aussi des versions indépendantes d'UNIX, non basées sur la license AT&T:

LINUX	Linus Torvalds	Domaine Public
GNU	Free Software Foundation – Richard Stallman (MIT)	Domaine public

- Plusieurs efforts de standardisation entre constructeurs sont en cours: consortium OSF, norme POSIX (P1003), X/OPEN.

La structure générale d'UNIX

- **un noyau**. Transparent à l'utilisateur. Gère les process, la mémoire et les fichiers. Facile à porter sur un nouveau matériel.
- **des fonctions système**. Disponibles pour la programmation système: gestion d'entrées/sorties, interruptions, horloges, ...
- **Une librairie de fonctions**. Disponibles pour la programmation courante: entrées/sorties C, date/heure, chaînes de caractères.
- **des utilitaires**. Commandes UNIX: éditeurs de texte, compilateurs, manipulation de fichiers, messageries, outils de génie logiciel, utilitaires réseaux, etc ...
- **des fichiers**. Notion très générale sous UNIX: même la mémoire physique est vue comme un fichier. UNIX gère les fichiers sous la forme d'une structure arborescente, fondée sur des répertoires de fichiers (directory).

Le rôle du langage C dans le système UNIX

- Le noyau d'UNIX est écrit principalement en C (et partiellement en assembleur). Les utilitaires sont tous écrits en C.
- Les appels systèmes et autres fonctions de librairies sont écrits pour être appelés à partir de programmes en C.
- Un compilateur C est en général le seul compilateur fourni avec tout système UNIX.
- La quasi totalité des logiciels du domaine public sont disponibles sous forme de programmes C qui doivent être compilés.
- Certains utilitaires UNIX utilisent des langages proches du C (c-shell, awk, perl).
- **Mais**:
  - ▷ On peut utiliser d'autres langages sur un système UNIX
  - ▷ On peut programmer en C sur des systèmes non UNIX

■ Une session UNIX

- **connection**: Prompt d'identification, fenêtre d'accueil X11, commande `ssh` (`telnet` et `rlogin` ne sont pas sécurisés).
- **identification**:
  connection: `ssh` *username*`@`*hostname*
  `password`: *mot-de-passe*
  Le système exécute un shell, puis le contenu des fichiers `.cshrc` et `.login`, puis attend des commandes après avoir affiché un prompt (`>`, `%`, `NOM-DE-STATION >`, ...).
- **notion de shell**: gestionnaire des commandes entrées par l'utilisateur.
  Associé à un langage de développement de "*scripts*"
  Il existe de nombreux shells: **csh**, **sh**, **bash**, **ksh**, **tcsh**,...
- **premières commandes. options, arguments**:
  affichage de la liste des fichiers du répertoire courant:
  `ls`
  `ls -la` *nom-de-fichier*
  `ls -l -a`
- **changer son password**: commande **passwd**
  Il faut ensuite taper son mot de passe courant, puis deux fois le nouveau.

  Le mot de passe constitue l'unique rempart contre des accès frauduleux à un ordinateur. Il est indispensable de respecter les quelques règles suivantes:

  - ▷ Un mot de passe est une chaine quelconque de 8 caractères maximum. Il faut utiliser au moins 6 caractères.
  - ▷ Il faut éviter d'utiliser des mots simples: mots courants, même en langues étrangères, prénoms, noms propres en général. Par confrontation à des dictionnaires de mots, il est très facile de *craquer* de tels mots de passe.
  - ▷ Un bon mot de passe est fait d'un mélange de lettres et de chiffres, d'un mélange de majuscules et minuscules, de suites de premières lettres des mots d'une phrase facile à retenir, de mots inventés ...
  - ▷ Ne jamais donner son mot de passe à quelqu'un. C'est aussi une question de responsabilité personnelle: en cas d'utilisation frauduleuse d'un compte, il peut être difficile de prouver que ce n'est pas le propriétaire qui s'en servait au mauvais moment.
- **déconnection**: `logout`, `exit`, `^D`
  Le système exécute le contenu du fichier `.logout`, puis termine la session.

Le Help en ligne

- Commande:
  man [*section*] { *commande*| *fonction* }
  man [`-s` *section*] { *commande*| *fonction* }
- Huit sections:
  1 Commandes standards
  2 Fonctions système en librairie C
  3 Librairie des fonctions C
  4 Description des fichiers spéciaux et autres drivers
  5 Structure des fichiers systèmes
  6 Jeux
  7 Traitement de texte et PAO
  8 Commandes systèmes
- Les commandes **apropos**, **whatis** ou **man -k** permettent de retrouver les commandes associées à un certain sujet.
- Les commandes internes à un shell (commandes *built-in*: voir page 31) sont à trouver dans les man-pages du shell correspondant "`man csh`", par exemple.
- Dans la documentation UNIX, les commandes et fonctions sont toujours citées avec leur numéro de section entre parenthèses. Exemple: `ls(1)`.

La structure des fichiers

Répertoires et inodes

Deux éléments contiennent la description des fichiers: les répertoires (ou directory) et les inodes.

- un **répertoire** est un tableau de couples *nom de fichier* et *numéro d'inode*
- un **nom de fichier** est une chaine de 255 caractères maximum
- Un **inode** est un enregistrement contenant toutes les informations concernant un fichier:
  les protections, les liens, le propriétaire, le groupe, la taille, les dates de dernier accès, de dernière modification et de dernière modification de l'inode, plus quelques informations spécifiques aux fichiers spéciaux.
- un **numéro d'inode** est un index dans un tableau d'inodes

- **Répertoires importants**:
  - ▷ répertoire racine (*root directory*): "/"
  - ▷ répertoire courant: "`.`"
  - ▷ répertoire père: "`..`"
  - ▷ le répertoire principal d'un utilisateur (*home directory*): `~`, `~`wenger, `$HOME`
  - ▷ répertoires système: `/bin`, `/usr/bin`, `/etc`, `/usr/man`, ...
  - ▷ répertoires temporaires: `/tmp`, `/usr/tmp`, ...

- **Commandes de base**:

ls -d	Affiche les répertoires (et pas les fichiers qu'ils contiennent).
mkdir *nom-dir*	Création d'un répertoire
rmdir *nom-dir*	Suppression d'un répertoire. Il ne doit plus contenir de fichiers
pwd	Affiche le répertoire courant
cd [*nom-dir*]	Changement de répertoire. Par défaut: répertoire principal de l'utilisateur.

- **Commandes de navigation dans les répertoires**:
  Les commandes suivantes permettent de mémoriser l'historique des répertoires courants en plaçant, avant chaque changement, le répertoire courant au sommet d'une pile, afin de pouvoir y retourner.

`pushd` [*nom-dir* | *+n*]	Met le répertoire courant sur le dessus de la pile et rend le nouveau répertoire courant. Sans arguments, échange les deux répertoires sur le sommet de la pile `+n` échange le dessus de la pile avec le $n^{ème}$ élément de la pile.
`popd` [*+n*]	Supprime le dessus de la pile des répertoires et rend courant le nouveau sommet. `+n` place au sommet et rend courant le $n^{ème}$ élément de la pile.
`dirs`	Affiche la pile des répertoires

Fichiers

- **Nom**:

  ▷ Limité à 255 caractères.

  ▷ Tout caractère est accepté. Eviter `?`, `*`, `[ ]`, { } et l'espace (blanc) car ces caractères jouent un rôle particulier dans la désignation de fichiers. D'une façon plus générale, il est préférable de se limiter aux caractères suivants: `A...Z`, `a...z`, `0...9`, `_ - . + , =`

  ▷ La commande `ls` n'affiche pas par défaut les fichiers commençant par un point. Il faut utiliser l'option `-a`.

  ▷ Il existe de nombreuses conventions de désignation de fichiers, utilisant une extension sous la forme "`.ext`":

.c	Source C	.tex	Source TEX ou LATEX
.h	Header C (include)	.dvi	Device independant file (résultat TEX ou LATEX)
.o	Module objet	.tar	Archive obtenue par `tar`
.f	Source Fortran	.Z	Fichier comprimé par `compress`
.C,cc,cpp,cxx	Source C++	.zip	Fichier comprimé par `zip/unzip`
.java	Source Java	.gz	Fichier comprimé par `gzip`
.class	Classe Java compilée	.shar	Shell Archive
.a	Librairie statique	.ps	Fichier Postscript (Adobe)
.so	Librairie partageable	.pdf	Fichier PDF (Adobe)
.y	Source yacc	.html, .htm	Fichier HTML (Web)
.l	Source lex	.gif	Image au format GIF
.s	Source assembleur	.jpg	Image au format JPEG
.pl	Source PERL	.png	Image au format PNG
		.fits	Fichiers au format FITS

- **Types de fichier**:
  - ▷ Fichiers ordinaires:
    - ⋆ Fichiers de données:
      - · ASCII (0–127), ASCII imprimable (32–126), binaire (0–255)
      - · fichiers de textes: caractères ASCII imprimables et un caractère "*newline*" (`'\n'`) en fin de ligne.
    - ⋆ Fichiers exécutables
  - ▷ Répertoires
  - ▷ Fichiers spéciaux, représentant un périphérique
    Un cas particulier utile est le fichier `/dev/null` qui peut être utilisé comme fichier vide en entrée, et comme "*trou noir*" en sortie.
  - ▷ Lien symbolique: pointeur vers un autre fichier
- **Désignation d'un fichier**:
  - ▷ Par son nom dans le répertoire courant: `monprog.c`
  - ▷ De façon absolue: `/usr/users/wenger/bin/monprog`, `~wenger/bin/monprog`
  - ▷ De façon relative au répertoire courant: `bin/monprog`, `../src/monprog.c`
- **Protection d'un fichier**:
  - ▷ **accessibilité** au propriétaire (`u`), au groupe (`g`) et aux autres (`o`)
  - ▷ **autorisation** de lire (`r`), d'écrire et supprimer (`w`), d'exécuter (`x`)
  - ▷ La protection d'un fichier s'écrit `rwxrwxrwx`. Ce qui peut se traduire par un nombre octal de 3 chiffres. Exemple: `644 = 110 100 100` ⇒ `rw.r..r..`
  - ▷ La protection d'un fichier peut être modifiée par la commande **chmod**.
  - ▷ La protection par défaut est définie par la commande "`umask` *valeur*"
    valeur est un nombre octal de 3 chiffres dans lequel les bits à 1 correspondent aux protections annulées. Quelques valeurs utiles:
    - ⋆ umask 022 – interdit l'accès en écriture au groupe et aux autres (`rwxr.xr.x`)
    - ⋆ umask 027 – interdit l'écriture au groupe et tout aux autres (`rwxr.x...`)
- **Particularités des protections lorsqu'elles concernent un répertoire**:
  - ▷ `x` permet l'accès à un répertoire: Commandes `cd`, `pwd` et utilisation dans un *path*
  - ▷ `r` permet la lecture du contenu du répertoire: Commande `ls`
  - ▷ `x` sans `r` permet la référence à un répertoire, mais pas le listage de son contenu
  - ▷ `w` permet la création et la suppression de fichiers dans le répertoire
- **Commandes de gestion des fichiers**:

ls -lF Affiche les principales caractéristiques d'un fichier
Exemple:

```
 -rw-r--r-- 1 wenger   simbad        978  Nov 5 1990  monprog.c
 -rwxr-x--- 1 wenger   simbad      37654  Nov 5 1990  monprog*
(a)  (b)   (c) (d)       (e)         (f)     (g)          (h) (i)
```

(a) Type de fichier:
    -: fichier, d:répertoire, l:lien, b/c:device, s:socket
(b) 3 series de rwx: protections propriétaire, groupes et autres
(c) Nombre de liens hard
(d) Nom du propriétaire
(e) Nom du groupe
(f) Taille du fichier (en octets)
(g) Date de dernière modification du fichier
(h) Nom du fichier
(i) Indicateur de type, mis en place par l'option F:
    *: exécutable /: répertoire @: lien =: socket

Options utiles:
-l Listage détaillé
-F Affiche le type pour certains fichiers
-g Liste aussi les noms de groupe
-a Liste aussi les fichiers commençant par un point
-d Liste les caractéristiques d'un répertoire, pas le contenu
-R Liste récursivement tous les sous-répertoires

wc [-lwc] Affiche le nombre de lignes, de mots et de caractères d'un fichier

cat *fic1 fic2...* Concaténation de fichiers.
`cat fich1 fich2 > FICH` concatène `fich1` et `fich2` dans FICH

cat *fich* Affiche le fichier à l'écran
Quelques options utiles:
-b supprime les lignes blanches redondantes
-v affiche de façon visible les caractères de contrôle
-t affiche de façon visible les caractères de tabulation

more *fich* Affichage contrôlé d'un fichier à l'écran
Les commandes principales sont:
<RET>: avance d'une ligne
<espace>: avance d'une page
"q": quitte
more dispose de nombreuses commandes pour se déplacer dans le fichier:

<espace>, z	Affiche le prochain écran
<RET>	Déplace le texte d'une ligne
q, Q	Sortie du programme
d	Déplace l'affichage d'un demi-écran (11 lignes)
kf	Avance de k écrans de texte
kb,^B	Recule de k écrans
/<ExprReg>	Recherche l'expression régulière (cf p 39)
n	Recherche la prochaine occurence de l'expression régulière
v	Démarre vi à la ligne courante
.	Répète la commande précédente
^L	Reaffiche l'écran
=	Affiche le numéro de la ligne courante
:f	Affiche le nom du fichier et le numéro de la ligne
!<cmde unix>	Exécute la commande unix dans un sous-shell
:n	Va au fichier suivant
:p	Va au fichier précédent

cp [-pr] *f1 f2* Copie *f1* sur *f2*
cp opt *f... dir* Copie les fichiers dans le répertoire désigné
-p conserve les dates courantes
-r copie aussi les sous-répertoires de façon récursive

mv *fich1 fich2* Modifie le nom d'un fichier dans un répertoire.
mv *fich... dir* Déplace le nom des fichiers spécifiés dans le répertoire désigné.

- ▷ Est utilisé pour renommer un fichier dans le même répertoire
- ▷ Utilisé entre deux répertoires dans le même *file systeme* (cf p 20), seul les répertoires seront modifiés. L'opération est très rapide.
- ▷ Utilisé entre deux répertoires dans des *file systems* différents, les fichiers eux-mêmes sont entièrement recopiés dans le nouveau *file system*.
- ▷ mv peut aussi renommer, ou déplacer un répertoire. Mais uniquement s'il reste dans le même *file system*.

rm *fich* Suppression d'un fichier
A condition que ce soit le dernier lien hard (cf p 20).

chmod *prot fich* Change les protections d'un fichier (cf p 16)
chmod {u|g|o|a} {+|-|=} {r|w|x} *fich*
+: ajouter, -: supprimer, =: remplacer les protections existentes

file *fich* Affiche le type de fichier, en fonction de son contenu

du Affiche la taille des répertoires d'une arborescence.
Cette commande permet à un utilisateur de contrôler la place qu'il occupe sur disque, par répertoire.

df Liste les *file system* (cf p 20) et leur taux de remplissage.
L'option -i donne aussi le taux de remplissage des tables d'inodes.

find *dirlist crit...* Recherche des fichiers dans une liste de répertoires en fonction de différents critères.
La liste des critères est parcourue tant que les précédents sont vrais
Exemple: `find / -name monprog -print`
Affiche tous les fichiers débutant par *monprog* sur l'ensemble des disques d'une station (départ de la recherche à "/" = répertoire racine)

Liste des critères de recherche principaux:

`-name` *fich*	Vrai si le fichier correspond à *fich*. Tous les *wildcards* (cf p 34) peuvent être utilisés.
`-print`	Toujours vrai. Imprime le nom complet du fichier courant.
`exec` *cmde*	Vrai si la commande retourne un statut égal à 0. Dans la commande, le nom du fichier courant peut être désigné par {}. La fin de la commande doit être marquée par \; Ex: `find ~ -name core -exec rm {} \;`
`-type` *t*	Vrai si le type du fichier est *t*, avec *t* = `f`:fichier, `d`:répertoire, `l`:lien symbolique
`-perm [-]` *perm*	Vrai si le fichier à les permissions spécifiées *perm* doit être donné sous forme octale Ex: *-perm 0222* signale les fichiers que tout le monde peut écrire `[-]` permet de tester seulements certaines permissions
`-user` *uname*	Vrai si le fichier appartient à *uname*
`-group` *gname*	Vrai si le fichier appartient au groupe *gname*
`-size` $\pm n$	Vrai si le fichier à la longueur n (`n`), fait moins de n caractères (`-n`) ou plus de n caractères (`+n`)
`-atime` $\pm n$	Vrai si le fichier a été accédé il y a n jours (`n`), il y a moins de n jours (`-n`), ou n'a pas été accédé depuis au moins n jours (`+n`).
`-mtime` $\pm n$	Idem `atime` avec la date de dernière modification des données du fichier
`-ctime` $\pm n$	Idem `atime` avec la date de dernière modification des caractéristiques du fichier (modification de l'inode)
`-mount`	Vrai si le fichier courant est sur le même *file system* que le répertoire courant dans *dirlist*.
\(*crit1* **-o** *crit2* \)	OU logique sur des critères
`!`*crit*	Négation d'un critère Ex: `find ~wenger \! \( -user wenger -o -group simbad \) -type d -print`

- **Liens entre fichiers**:
  - ▷ **Liens hard** (*hard links*):
    - ⋆ Un lien hard est une entrée normale dans un répertoire, pointant sur le même *inode*, donc la même description de fichier.
    - ⋆ Il n'est pas différent d'une autre entrée dans le répertoire.
    - ⋆ Strictement rien ne distingue un lien hard du nom initial d'un fichier.
    - ⋆ Il ne peut être utilisé que pour des fichiers normaux sur le même *file system*.
    - ⋆ Il est crée par la commande: **ln** *fich1 fich2*
      fich2 est un lien hard vers le fichier *fich1*
    - ⋆ Il est supprimé par la commande: `rm` *fich1*
      Le lien *fich1* est supprimé. Le fichier existe encore sous son nom *fich2*. La suppression du dernier lien supprime aussi le fichier.
  - ▷ **Liens symboliques** (*Symbolic links*):
    - ⋆ Un lien symbolique est un nouveau fichier crée dans un répertoire. Le contenu du fichier est le nom d'un autre fichier vers lequel il pointe.
    - ⋆ Un lien symbolique peut être vu comme un pointeur indirect sur un fichier.
    - ⋆ Il est traité comme un lien hard par la plupart des programmes.
    - ⋆ Il peut s'appliquer à tout type de fichier, y compris des répertoires et des fichiers sur d'autres *file systems*.
    - ⋆ Il est crée par la commande: **ln –s** *fich1 fich2*
      fich2 est un lien symbolique vers *fich1*
    - ⋆ Il est supprimé par la commande: `rm` *fich2*
      Supprimer le fichier réel, sans supprimer le lien, n'est pas interdit, mais va poser des problèmes lors de références ultérieures au lien.

Disques – Système de fichiers – File systems

- Un disque dur peut être découpé en huit **partitions** (**a**–**h**) constituant autant de *systèmes de fichiers* indépendants.
- Un **file system** est une arborescence de répertoires et de fichiers constituant un disque complet ou une partition d'un disque.
- Le *file system* initial constitue la partition "**a**" du disque système et a pour répertoire de départ "/"
- Tout autre *file system* constituant un disque ou une partition d'un disque est lié à un *file system* existant au moyen de la commande `mount`. Par cette commande, le répertoire racine du nouveau *file system* devient un répertoire dans le *file system* existant.
- Tout fichier doit être entièrement contenu dans un *file system* et un seul. La commande `df` permet de connaitre le taux de remplissage des différents *file system* montés.

Les éditeurs de texte

- Plusieurs éditeurs de texte sont disponibles sous UNIX:
  - ▷ **vi**: toujours disponible, universel, interface pleine page de l'éditeur ligne à ligne **ex**.
  - ▷ **emacs**: domaine public, puissant, extensible grâce à un langage de programmation.
  - ▷ **nedit**: très puissant, utilisable sur terminal X.
  - ▷ **axe**: utilisable sur terminal X.
  - ▷ **xedit**: limité, utilisable sur terminal X.
  - ▷ **kedit**: fournit avec l'environnement `KDE` sous Linux.
- Un éditeur de texte n'est pas un logiciel de traitement de texte et encore moins un logiciel de PAO.
- Les principales fonctions de tout éditeur de texte sont:
  - ▷ Déplacement dans le fichier et déplacement du curseur dans la fenêtre à l'écran.
  - ▷ Insertion et suppression de texte
  - ▷ Recherche et remplacement de texte
  - ▷ Copie et déplacement de sections de texte
  - ▷ Lecture et écriture de fichiers

■ L'éditeur vi

Introduction

- **vi** à été conçu pour UNIX, dans un esprit d'indépendance vis à vis de tout type de matériel (terminal et clavier).
- **vi** est appelé par la commande:
  - ▷ `vi [-r] fichier ...`
  - ▷ L'option **–r** permet en général de récupérer un fichier avec (presque) toutes ses modifications après un plantage pendant la mise à jour de ce fichier. A condition de **rappeler ce fichier par** `vi` **en tout premier après le plantage**.
  - ▷ `view fichier` permet d'accéder à un fichier en mode lecture seulement (*read only*).
- "~" montre les lignes au-delà de la fin du fichier
- La dernière ligne de l'écran est une ligne de statut:
  Ex: `"monprog.c" [Modified] line 60 of 259 --23%--`
  La commande `^G` affiche cette ligne de statut.
- Toutes les commandes de vi sont des séquences de caractères alphabétiques, numériques ou caractères de ponctuation normaux. vi peut se passer de touches spéciales pouvant exister sur un clavier: flèches de déplacement de curseur, suppression de caractère, etc ...
- Comme ces mêmes caractères servent à saisir le texte dans l'éditeur, **vi** possède deux modes de fonctionnement:
  - ▷ **le mode commande** dans lequel seront exécutées toutes les commandes d'édition: déplacements, suppression de caractères, recherche de texte, etc ...
  - ▷ **Le mode insertion** pour la saisie de texte

  Aucun affichage particulier n'indique dans quel mode on est.
- `vi` **démarre en mode commande**
- **le mode insertion** est démarré, entre autre, par la commande "**i**", qui permet l'insertion de caractères avant le curseur.
- On retourne en mode commande par `<ESC>`.
- De nombreuses commandes peuvent être précédées d'un entier n pour répéter la commande n fois.
- Les commandes commençant par "**:**" sont des commandes de l'éditeur **ex** sous-jacent à **vi**.
  Ces commandes apparaissent sur la ligne de statut de l'éditeur (dernière ligne de l'écran).
- Les commandes commençant par / ou ? concernent les recherches de texte.
- La **sortie de l'éditeur** se fait par une des commandes suivantes:

  :wq, **ZZ** sauvegarde du fichier et sortie
  :q! sortie sans sauvegarde des modifications

- Un texte peut être considéré comme constitué de:
  - ▷ **lettres** (caractères)
  - ▷ **mots**, qui peuvent être délimités seulement par des blancs ou, de façon plus restrictive, par n'importe quel caractère de ponctuation.
  - ▷ **phrases**, délimitées par un point
  - ▷ **paragraphes**, délimités par une ligne blanche
  - ▷ **sections**, notion liée au type de fichier édité.

  Un texte peut aussi être vu comme fait de **lignes** et **d'écrans**.
  La plupart des fonctions d'édition ont des commandes pouvant s'appliquer à tous ces niveaux.

Déplacement du curseur dans la fenêtre

h	déplace le curseur vers gauche
j	déplace le curseur vers le bas
k	déplace le curseur vers le haut
l, <espace>	déplace le curseur vers la droite
H	va à la première ligne de l'écran
M	place le curseur sur la ligne du milieu de l'écran
L	va à la dernière ligne de l'écran
0	place le curseur en début de la ligne courante
^	place le curseur sur le premier caractère non blanc de la ligne courante
$	place le curseur en fin de ligne
k|	Déplace le curseur à la colonne k de la ligne courante
<ret>,+	Avance au début de la ligne suivante
−	Recule au début de la ligne précédente
w	avance au début du mot suivant, stoppé par la ponctuation
W	avance au début du mot suivant, arrêté seulement par un blanc
b	recule au début du mot précédent, délimité par la ponctuation
B	recule au début du mot précédent, délimité par un blanc
e	avance à la fin du mot suivant, hors ponctuation
E	avance à la fin du mot suivant, ponctuation incluse
tcar	Positionne le curseur avant le prochain caractère car, dans la ligne courante
Tcar	Positionne le curseur après le précédent caractère car, dans la ligne courante
fcar	Recherche dans la ligne le prochain caractère car
Fcar	Recherche dans la ligne le précédent caractère car
;	Répète la précédente commande f ou F
,	Répète la précédente commande f ou F, en inversant le sens
%	Recherche du pendant à un caractère ([{ ou)]}: Si le curseur est positionné sur une parenthèse ouvrante, % ira le positionner sur la parenthèse fermante correspondante, etc ...
(	recule le curseur au début de la phrase
)	avance le curseur au début de la phrase suivante
{ }	même comportement que "()" pour des paragraphes
[[ ]]	même comportement que "()" pour des sections

Déplacement de la fenêtre dans le texte

:n, nG	Va à la ligne n. Em particulier 1G va au début du texte
G	Va à la fin du fichier
^F	avance d'une page
^B	recule d'une page
^U	avance d'une demi-page
^D	recule d'une demi-page
^Y	avance d'une ligne, le curseur restant sur sa ligne
^E	recule d'une ligne, le curseur restant sur sa ligne
z<RET>	Met la ligne du curseur au haut de l'écran
z.	Met la ligne du curseur au milieu de l'écran
z-	Met la ligne du curseur au bas de l'écran
^L	Réaffiche l'écran. Utile lors de superpositions de messages systèmes à l'écran

Insertion de texte

Plusieurs commandes permettent de passer en mode **insertion de texte**.
Une seule commande permet de retourner en mode commande: <ESC>

i	Insère le texte **avant le curseur**
I	Insère le texte au début de la ligne
a	Insère le texte après le curseur
A	Insère le texte **à la fin de la ligne**
o	Insère le texte sur une nouvelle ligne ouverte **après la ligne courante**
O	Insère le texte sur une nouvelle ligne ouverte avant la ligne courante
^V	Permet de saisir un caractère de contrôle

Passage en mode commande

<ESC>	Passage du mode insertion en mode commande Taper <ESC> alors que l'on est en mode commande provoque un signal sonore. C'est le meilleur moyen de s'assurer que l'on est en mode commande. <ESC> a aussi pour effet d'annuler une commande en cours de saisie.

Remplacement de texte

rcar	Remplace le caractère courant par car. Reste en mode commande
R	Remplace le texte écrit jusqu'au prochain <ESC>
s	Remplace le caractère courant par un nouveau texte de longueur quelconque. Préfixé par un nombre n, le remplacement concernera n caractères
S	Remplace toute la ligne courante par un nouveau texte
cc	idem **S**
C	Remplace tous les caractères jusqu'à la fin de la ligne
~	Transforme les caractères de maj en min. et reciproquement

c<deplac^t>	remplace tous les caractères entre le curseur et le résultat du déplacement. Ce déplacement peut être: `w,W,b,B,0,^,$,(,),{,},%,nG,<RET>,+,-,H,M,L,'a,/ExprReg...` Exemples: `c$` = `C`, remplacement jusqu'à la fin de la ligne `cw` : remplacement jusqu'à la fin du mot courant `c3w` : remplacement jusqu'à la fin du $3^{ème}$ mot

Par des commandes **ex**, il est possible de faire des remplacements de texte globaux sur tout ou une partie du fichier:

`:s/ExprReg/text/`	Remplace la première occurence de l'expression régulière (cf p 39) par le texte dans la ligne courante
`:s/ExprReg/text/g`	Remplace toutes les occurences de l'expression régulière par le texte dans la ligne courante
`:n1,n2s/ExprReg/texte/g`	Remplace les occurences de l'expression régulière dans les lignes $n1$ à $n2$.
`:g/ER1/s/ER2/text/g`	Dans toutes les lignes contenant l'expression regulière `ER1`, remplacer l'`ER2` par text
`:n1,n2g/ExprReg/cmde-ex`	Execute la commande ex dans toutes les lignes entre $n1$ et $n2$ et contenant l'expression régulière Ex: `:g/##/d` – supprime toutes les lignes contenant `##`

Suppression de texte

x	supprime le caractère sous le curseur
X	supprime le caractère à gauche du curseur
dd	supprime la ligne courante
D	supprime les caractères jusqu'à la fin de la ligne
J	supprime le saut de ligne entre la ligne courante et la suivante
d<deplac^t>	supprime les caractères entre le curseur et le résultat du déplacement. Ce déplacement peut être: `w,W,b,B,0,^,$,(,),{,},%,nG,<RET>,+,-,H,M,L,'a,/ExprReg...` Exemples: `d$` = `D`, suppression jusqu'à la fin de la ligne `dw` : supprime les caractères jusqu'à la fin du mot courant `d%` : placé sur une parenthèse ou un crochet ouvrant, supprime les parenthèses ou les crochets et leur contenu
:*pos1*[,*pos2*]**d**	Supprime les lignes entre *pos1* et *pos2* (commande **ex**) *pos1* et *pos2* peuvent être: Des numéros de ligne: `:12,35d` Des nombres de lignes: $+3$ (3 prochaines), -4 (4 dernières) La fin du fichier: `$` La ligne courante: `.` Des expressions: `$-10` (les 10 dernières lignes)

Annulation d'une commande

u	Annule l'effet de la dernière commande réalisée.
U	Remet la ligne courante dans son état initial avant modifications. Cette commande n'est valable que tant que le curseur n'a pas changé de ligne.

Répétition d'une commande

. (point)	Répète la dernière commande effectuée

Marquage de texte

On peut définir dans **vi** 26 marques dans le texte désignées par les lettres minuscules **a** - **z**.
Les commandes suivantes sont associées à ce marquage:

mcar	défini la position courante du curseur comme étant la marque car
‘car	va à la marque car (‘ = accent grave, *backquote*)
’car	va au début de la ligne contenant la marque car (’ = apostrophe simple, accent aigu, *quote*)
‘‘	va à l'endroit d'où est parti le dernier déplacement
’’	va au début de la ligne d'origine du dernier déplacement

Mouvements de texte (couper/coller)

- Il y a deux types de mouvements de textes:
  - ▷ Le déplacement: on supprime le texte de son emplacement initial et on le place ailleurs.
  - ▷ La recopie: on laisse le texte dans son emplacement initial et on en place une copie ailleurs.

  Le principe consiste à placer la zone à déplacer ou à copier dans un buffer et à la reécrire dans son nouvel endroit après déplacement du curseur.
- **vi** gère un certain nombre de buffers, qui sont autant de zones indépendantes pouvant contenir du texte:
  - ▷ **un buffer fichier** qui contient le fichier qui a été lu et qui est en cours d'édition
  - ▷ **des buffers de texte**:
    - ⋆ 26 buffers à usage général nommés **a** - **z**
    - ⋆ 1 buffer (sans nom) reçevant par défaut les textes supprimés.
    - ⋆ 9 buffers, numérotés de **1** à **9**, sauvegardant le résultat des 9 dernières suppressions
- Les mouvements se font au moyen de trois commandes:

d<deplac^t**>**	place le texte compris entre le curseur et le résultat du déplacement du *buffer fichier* dans un *buffer texte* et supprime le texte origine (dans le buffer fichier). Le déplacement peut être: `w,W,b,B,0,^,$,(,),{,},%,nG,<RET>,+,-,H,M,L,'a,/ExprReg...`
y<deplac^t**>**	recopie le texte compris entre le curseur et le résultat du déplacement du *buffer fichier* dans un *buffer texte*. Le déplacement peut être: `w,W,b,B,0,^,$,(,),{,},%,nG,<RET>,+,-,H,M,L,'a,/ExprReg...`
p, **P**	placent le texte d'un *buffer texte* dans le *buffer fichier* après (**p**) ou avant (**P**) la ligne courante.
dd	supprime la ligne courante
yy	recopie la ligne courante

- Par défaut, tous les transferts se font avec le buffer par défaut.
- Si une de ces commandes est précédée de `"nom-de-buffer`, celui-ci remplace le buffer par défaut. Exemples:

"xd'a	met dans le buffer `x` le texte entre le curseur et la marque `a`.
"xp	dépose le contenu du buffer `x` à l'endroit du curseur dans le fichier.
"bdw	supprime la fin du mot courant et range ces caractères dans le buffer **b**
"ayy	sauve la ligne courante dans le buffer **a**
"*n*p	met dans le *buffer fichier* le contenu de la $n^{ème}$ suppression

- Si un buffer est désigné par une majuscule (`A - Z`), alors le texte sera **rajouté** à la fin du buffer.

Commandes associées aux fichiers

- Lecture de fichiers:

vi *fich*	`vi` lit automatiquement le fichier en argument sur la ligne de commande
:r $fich$	Insère le contenu du fichier après la ligne courante

- Ecriture de fichiers:
  Ces commandes sont les commandes de **sauvegarde** d'un fichier.

:w	Ecrit le texte dans le fichier courant sur disque **Il est bon d'utiliser cette commande régulièrement pendant la saisie d'un texte long**
:w $fich$	Ecrit le texte dans le fichier nommé
:w! $fich$	Ecrit le texte dans le fichier nommé, si celui-ci existe déjà
:i, j**w** $fich$	Ecrit les lignes i à j du texte dans le fichier $fich$
:wq, **:x**	Ecrit le texte dans le fichier sur disque et **sort de l'éditeur**

- Insertion du résultat de commandes unix:

:r !*cmde unix*	Insère le résultat de la commande dans le fichier

- Fonctionnement multifichiers:
  vi peut charger plusieurs fichiers lors de l'appel: `vi fich1 fich2 ...`
  On peut ensuite en changer au moyen des commandes suivantes:

:e *fich*	Charge le fichier fich
:n	Rend courant le fichier suivant
:e#, `ctrl-^`	switch entre deux fichiers

Recherche de texte

- Les commandes:

/$text$`<ret>`	Recherche de $text$ en avant. Rebouclage à la fin du fichier
?$text$`<ret>`	Recherche de $text$ en arrière avec rebouclage au début du fichier
n	Recherche le prochain texte, dans la même direction
N	Recherche le prochain texte, dans la direction opposée
`/<ret>`	Recherche le prochain texte, toujours vers l'avant
`?<ret>`	Recherche le prochain texte, toujours vers l'arriere

- Ces commandes de recherche de caractères dans la ligne courante sont aussi des commandes de positionnement de curseur et peuvent être utilisées combinées à des commandes de suppression, de modification et de recopie de texte.
  Ex: `dfx` supprime les caractères entre le curseur et le prochain caractère `x` dans le même ligne.

- Les recherches de textes sont par défaut étendues aux expressions régulières (cf p 39). Ceci peut être supprimé par la variable: "*:set nomagic*".

Commandes de modifications diverses

Ces commandes ne sont pas particulières, mais sont des combinaisons de commandes élémentaires. Leur intérêt est d'être d'un usage courant:

xp	échange le caractère courant et le suivant
ddp	échange la ligne courante avec la suivante
yyp	duplique une ligne
`dwwP`	échange le mot courant et le suivant
`xPP`	double un caractère
`bdw`	supprime le mot sur lequel est le curseur, quelle que soit sa position.
`ea`	rajoute du texte à la fin du mot courant
`bi`	rajoute du texte au début du mot courant

Variables de l'éditeur

De nombreuses variables permettent de paramétrer le comportement de vi.

Les commandes suivantes permettent de les afficher:

`:set`	Affiche toutes les options qui ont une valeur différente de la valeur par défaut
:set all	Affiche toutes les options et leur valeur courante
`:set option?`	Affiche la valeur de l'option spécifiée

La modification d'une variable se fait par les commandes suivantes:

`:set [`*no*`]`*var*	Défini une variable logique. Le préfixe "*no*" la supprime
`:set` *var=val*	Affecte une valeur à une variable numérique.

Les principales variables sont:

`autoindent` (`ai`)	Indentation automatique (`^D` pour reculer d'une tabulation, `^T` pour avancer)
`ignorecase` (`ic`)	Ignore les maj/min lors de recherches
`number` (`nu`)	Affiche les numéros de ligne
`magic`	Autorise les expressions régulières dans les recherches de chaînes et les substitutions
`showmatch` (`sm`)	Montre la parenthèse ouvrante correspondant à une parenthèse fermante, lors de sa frappe, par un déplacement automatique du curseur.
`wrapscan` (`ws`)	Boucle à la fin du fichier pour continuer les recherches de chaînes
`tabstop` (`ts`) n	Met la tabulation à n caractères
`wrapmargin` (`wm`) n	Permet la saisie continue: vi passe automatiquement à la ligne si elle dépasse n caractères

La suppression d'une variable se fait par `set no`$variable$ (`set noai`) ou par la mise à zéro de sa valeur (`set wm=0`).

L'initialisation de l'éditeur peut se faire de deux façons:

- La définition d'une variable d'environnement EXINIT
  Ex: `setenv EXINIT "set ai sm ic"`
- La création d'un fichier `.exrc` dans le répertoire principal contenant les commandes désirées.

■ La PAO sous UNIX

Logiciels de type *markup language*:

- **TEX, LATEX: Logiciel domaine public, universellement utilisé en astronomie**. Ecrit par D. Knuth (Université de Stanford).
- **nroff/troff**: produit UNIX. Encore utilisé pour les *man-pages*.
- **eqn, tbl, pic**: filtres associés à `nroff` et `troff` pour les équations, les tables et les figures.

Logiciel de bureautique : **star office** (commande `soffice`).

Des fichiers **Postscript** et **Pdf** générés par de nombreux traitements de textes peuvent être affichés et imprimés sour Unix respectivement par les commandes `ghostview` (`gv`) et `acroread`.

L'utilisation des imprimantes

Commande d'impression

`lpr -P`*imprimante* `fichier ...`

`lp -d` *imprimante* `fichier ...`

Les types de fichiers

- **Les fichiers ASCII**:
  Ils s'impriment par défaut en format *portrait* jusqu'à 80 colonnes/ligne et en format *paysage* jusqu'à 132 colonnes/ligne (option `-r`)
- **Les fichiers TEX**:
  Ils doivent d'abors être compilés pour produire un *fichier dvi* (*device independant file*).
  La commande `prf` convertit automatiquement un fichier `.dvi` en Postscript avant de la sortir sur une imprimante qui doit donc être obligatoirement Postscript.
- **Les fichiers Postscript**:
  Ils peuvent être directement imprimés par la commande **prf**, mais exclusivement sur des imprimantes Postscript. Ils peuvent être lus avec la commande `ghostscript` et son interface `ghostview`.
- **Les fichiers Pdf**:
  Ils doivent être chargés dans `acroread` et imprimés depuis ce logiciel. Sous Linux, il existe plusieurs interprêtes de fichiers pdf (`kpdf`, `xpdf`).

Commandes associées à l'impression

- Commande de formattage d'un fichier:

pr Formattage d'un fichier pour l'impression. Cette commande permet de gérer la dimension des pages et des marges, de faire un fichier multi-colonnes et de rajouter des en-têtes sur chaque page.
Options de base:

`-w` n	Défini la largeur de la page
`-l` n	Défini la longueur de la page
`-o` n	Introduit une marge à gauche de n caractères
`-`n	Formatte en n colonnes. Les colonnes sont remplies verticalement
`-a`	Les colonnes sont remplies horizontalement
`-h` $texte$	Utilise $texte$ dans l'en-tête des pages, à la place du nom de fichier
`-t`	Supprime tout entête de page

- Commandes de contrôle de l'imprimante
  Ces commandes ne fonctionnent pas toujours, surtout si l'imprimante est ailleurs sur le réseau et pas directement connectée à la station. Un fichier en cours d'impression peut aussi être entièrement dans la mémoire de l'imprimante et échappera alors à ces commandes.

lpq	**–P***impr*	Affiche la liste des impressions en attente
lprm	**–P***impr* *jobid*	Permet de supprimer des impressions dans la file d'attente

Les shells – description du C-shell

Généralités

- Toute session interactive est contrôlée par un shell
- Le shell agit comme interface utilisateur avec le système
- Plusieurs programmes shell différents existent et ont des comportements différents:
  - ▷ **sh**: Bourne shell – shell originel d'Unix
  - ▷ **csh**: C-shell – évolution du Bourne shell avec une syntaxe plus proche du langage C
  - ▷ **bash**: Bourne Again shell – amélioration du Bourne shell
  - ▷ **ksh**: Korn shell – évolution du Bourne shell et du C-shell
- **Tout ce qui suit concerne le C-shell (csh)**
- Un programme shell est démarré à l'initialisation d'une session (*login*). Il exécute d'abord les commandes contenues dans le fichier `.login`, avant de donner la main à l'utilisateur. Il attend alors ses commandes, les interprête et les exécute.
- Une commande est toujours analysée comme une suite de mots
  - ▷ Le premier mot est la **commande**.
  - ▷ Les mots suivants constituent les **arguments**.

  Les premiers mots d'une commande (le nom et *n* arguments) peuvent être remplacés par un **alias**.
- Il y a deux types de commandes:
  - ▷ des **commandes internes** (*built-in*) exécutées par le shell lui-même. Exemples: `cd`, `exit`
  - ▷ des **commandes externes**: programmes exécutables et *scripts*. Exemples: `ls`, `cat`

Les commandes externes sont retrouvées automatiquement si elles sont dans des répertoires définis dans la variable d'environnement `PATH`. Elles peuvent aussi être écrites avec leur chemin absolu (`/bin/ls -l`) ou relatif (`~/bin/monprog`).
On peut obtenir la provenance d'une commande quelconque par la commande "**which** *cmde-unix*" qui retourne le chemin complet de la commande (sauf si c'est une commande interne).
Exemple: "`which ls`" affiche "`/bin/ls`" ou "`ls: aliased to 'ls -F'`" si la commande a été définie par un `alias` (cf p 38).

- Lors de l'exécution d'une commande:
  - ▷ Le process shell interactif est dupliqué par une fonction **vfork()**.
  - ▷ Un script sera interprété sous le contrôle d'un sous-shell (shell lancé par le shell courant).
  - ▷ Un programme sera chargé par une fonction **execve()**.
- Des commandes peuvent être incluses dans le shell par la commande:
  source *fichier-de-commandes*
- Plusieurs commandes peuvent être écrites sur une même ligne en les séparant par des point-virgules "**;**".
- L'exécution conditionnelle de commandes peut être réalisée par les opérateurs:
  `||` *cmde1* `||` *cmde2*
  cmde2 est exécutée seulement si *cmde1* se termine en erreur (statut de sortie différent de 0).
  `&&` *cmde1* `&&` *cmde2*
  cmde2 est exécutée seulement si *cmde1* se termine avec un statut de sortie égal à zéro

Caractères de contrôle

- La frappe des commandes peut être contrôlée par les caractères suivants:

`<DEL>`	efface les caractères entrés
`^W`	efface un mot
`^U`	efface la ligne complète
`^R`	reécrit le texte tapé (utile lors de l'affichage à l'écran de messages extérieurs)
`^V`	Permet la saisie d'un caractère de contrôle (Ex: `echo ^V^G` = *beep* sonore)

- Le déroulement des programmes interactifs peut être controlé par les commandes suivantes:

`^C`	Interrompt l'exécution du programme courant
`^Z`	Suspend l'exécution du programme courant (process stoppé) Son exécution peut être reprise par les commandes `fg` ou `bg` (cf p 36).
`^S`	Bloque l'affichage sur le terminal
`^Q`	Reprend l'affichage sur le terminal
`^O`	Interrompt l'affichage sur le terminal
`^D`	Envoi au programme, lisant ses données au clavier, le caractère *fin-de-fichier* (EOF)

Les fichiers standards. Redirection et pipes

- Tout programme UNIX a **trois fichiers standards**:

  ▷ **le fichier standard input** (*stdin*): pour l'entrée de données (descripteur n^o 0)
  Par défaut, entrée au clavier. Fonctions `scanf, getchar` en C.

  ▷ **le fichier standard output** (*stdout*): pour la sortie de résultats (descripteur n^o 1)
  Par défaut, sortie à l'écran. Fonctions `printf, putchar` en C.

  ▷ **Le fichier standard erreur** (*stderr*): pour la sortie des messages d'erreur (descripteur n^o 2)
  Par défaut, sortie à l'écran.

- Ces trois fichiers standards peuvent être **redirigés** depuis ou vers un autre fichier:

  ▷ `cmd <` *fich*
  Redirige le fichier standard input depuis le fichier *fich*

  ▷ `cmd >` *fich*
  Redirige le fichier standard output vers le fichier *fich*

  ▷ `cmd >>` *fich*
  Le fichier standard output est mis à la fin du fichier *fich*

  ▷ `cmd <` *fich1* `>` *fich2*
  Redirige *stdin* depuis *fich1* et *stdout* vers *fich2*

  ▷ `cmd >&` *fich*
  Redirige *stdout* **et** *stderr* vers *fich*

  ▷ l'autorisation d'écraser un fichier existant est contrôlée par la variable d'état **noclobber**:

  `set -o noclobber`: `cmd >` *fich* est interdit si le fichier *fich* existe déjà.
  Les opérateurs de redirection `>!`, `>>!`, `>&!` et `>>&!` permettent alors de forcer la redirection.

`set +o noclobber`: autorise à nouveau l'écrasement de fichiers existants.

▷ L'opérateur "`<< IFIN`" permet de rediriger en entrée les lignes qui suivent dans le shell, jusqu'à rencontrer une chaîne particulière spécifiée avec l'opérateur.
L'exemple suivant crée un fichier *fich* avec deux lignes de texte:

```
cat << %%FIN%% > fich
aaaaaaaaaaaaaa
bbbbbbbbbbbbbb
%%FIN%%
```

- Deux process peuvent **communiquer** par l'intermédiaire de leurs fichiers standards d'entrée et de sortie au moyen de **pipes**:

  ▷ L'opérateur "`|`" placé entre deux commandes envoie la sortie standard de la première commande en entrée standard dans la seconde.
  Ex: `ls | wc -l`
  La sortie de `ls` est envoyée en entrée dans la commande `wc -l`. La sortie de `wc -l` donnera le nombre de lignes de la commande `ls`, soit le nombre de fichiers dans le répertoire.

  ▷ Un *pipe* regroupant en sortie les fichiers *stdout* et *stderr* est obtenu par l'opérateur `|&`.
  Ex: `ls |& more`

  ▷ Redirections et pipes sont combinables:
  `cmd1 < fich1 | cmd2 |& cmd3 > fich2`

  ▷ La commande **tee** est exclusivement utilisée dans un *pipe*: Elle permet de rediriger son entrée standard à la fois dans un fichier et sur sa sortie standard:
  `... | tee fich.sauv | ...`

- La syntaxe des redirections et des pipes est différentes selon les shells. En bash on écrit:

`cmde >` *stdout*	redirection de la sortie standard
`cmde 2>` *stderr*	redirection de la sortie d'erreur
`cmde >` *stdout* `2>` *stderr*	redirection des deux sorties indépendamment
`cmde >` *fichier* `2>&1`	redirige stderr vers stdout et le tout vers un fichier. (mélange stdout et stderr dans une sortie unique)
`cmde 2>&1 | cmd2`	regroupe les deux sorties standards vers le pipe et la commande suivante

En plus de l'opérateur <<, il existe l'opérateur <<-: il permet d'indenter par des tabulations les lignes de texte à rediriger, mais sans que ces caractères de tabulation du début de la ligne ne soient transmis en standard input.

Substitution de noms de fichiers – Wildcards

- Certains caractères jouent un rôle particulier dans la définition des noms de fichiers:

	Défini le répertoire principal (*home directory*)
~*user*	Défini le répertoire principal de l'utilisateur spécifié
*****	Remplace n'importe quelle chaine de caractères (même vide) "`*.c`" désigne tous les fichiers qui se terminent par `.c` dans un répertoire (sauf "`.c`" lui-même).
?	Remplace n'importe quel caractère unique
[**abc...**]	Remplace n'importe quel caractère unique pris dans la liste. "`pgm.[cho]`" designe tous les `.c`, `.h` et `.o` d'un répertoire
[**! 0-9**]	N'importe que caractère sauf ceux de la liste
{ str, str, ... }	Désigne une quelconque des chaînes entre crochets Ex: `doc.{tex,log,aux}`

- Les différents caractères peuvent être combinés:
  Ex: `~wen*/{bin,src}/[a-z]*.{c,tex}`

- Tous les fichiers répondant à une telle expression sont retournés dans une liste triée alphabétiquement sur les noms.

- L'absence de fichier reconnu constitue une erreur qui termine la commande (message "`No match`").
  La définition de la variable de shell "`nonomatch`" évite de terminer en erreur ("`set nonomatch`")

- La variable de shell "`noglob`" interdit toute substitution
  Ceci permet de traiter des chaînes de caractères pour lesquelles il n'y a pas lieu de substituer des noms de fichiers.

Contrôle des process. Priorités

- Tout programme qui s'exécute est un process géré par le système

- L'utilisateur peut contrôler l'exécution de ses process au moyen de quelques commandes:

ps -ax	Donne la liste complète des process de l'utilisateur. Le deuxième champ est le **PID**, numéro du process. Ce nombre est nécessaire pour plusieurs commandes, dont `kill` ci-dessous.
^C	**Interrompt** le process interactif courant, mais cette interruption peut être *inhibée* par le programme.
^\	**Tue** le process interactif courant. (équivalent à `kill -9 PID`). Cette commande est toujours opérante car l'interruption générée ne peut pas être inhibée par le process.
^Z	**Stoppe** le process interactif courant. Voir ci-dessous page 36 le paragraphe consacré aux jobs, pour gérer un process stoppé.
kill -9 PID	**Tue** un process en donnant son numéro (PID = Process Identifier)
time *cmde*	Affiche après l'exécution de la commande diverses informations sur les ressources consommées: temps cpu, entrées/sorties, mémoire.

- Tout programme s'exécute avec une certaine priorité. Celle-ci est initialement la même pour tout programme.

- Selon l'unix utilisé, la priorité peut diminuer au bout de quelques minutes d'exécution si le programme est long.

- Un utilisateur peut lui-même diminuer la priorité d'un programme qu'il sait être long par la commande:
  nice [+prio] commande
  `prio` peut prendre des valeurs jusqu'à 20.
  Le nom de cette commande laisse entrevoir qu'il est agréable, pour les collègues sur la même station, de l'utiliser, surtout pour exécuter des programmes gros consommateurs de CPU. Si la station est peu chargée, l'exécution du programme ne sera pas ralentie. Dans le cas contraire, le programme long laissera plus de CPU aux éditeurs de texte et autres programmes interactifs.

- Après son lancement, un programme peut avoir sa priorité baissée par la commande:
  renice prio PID
  Ce qui permet de rattraper un oubli...

- Il est impossible à un utilisateur de remonter sa priorité.

Exécutions en tâche de fond – Jobs

- Toute commande terminée par un caractère "&" sera exécutée en **background**, c'est à dire en tâche de fond non interactive.
  Ex: `cmde > fich&`
  Une telle exécution ne doit avoir aucune interaction avec la fenêtre qui l'a lancée: ni saisie de données au clavier, ni affichage l'écran. S'il y a lieu, la sortie d'écran doit être redirigée dans un fichier.

- Au démarrage, le système lui attribue un numéro de job qui est affiché avec le numéro de process:
  `[1] 5528`

- A la fin de l'exécution, le système affiche sur le terminal d'où le programme a été lancé – s'il est ouvert – le message:
  `[1] Done cmde > fich`
  ou, s'il y a eu erreur:
  `[2] Exit 1 commande-en-erreur`

Par défaut, l'utilisateur ne recoit ce message qu'après la fin d'une commande interactive.
Il peut être prévenu immédiatement en définissant la variable `set notify`.
La commande "`notify %job`" permet d'être prévenu du changement de statut d'un job particulier.

- Un numéro de job est aussi attribué à un process stoppé par **^Z**.
- S'il reste des process stoppés lorsqu'on termine la session, le système affiche le message suivant à l'écran: "`There are stopped jobs`" , et la session reste ouverte. Une deuxième commande de fin de session terminera réellement celle-ci, en tuant les process stoppés.
- Les commandes suivantes permettent de manipuler les process:

jobs [-l]	Montre la liste des jobs actifs. L'option `-l` affiche aussi le PID
^Z	Stoppe un job interactif et lui attribue un numéro de job
bg %*job*	Exécute le job spécifié en batch (*background*)
fg %*job*	Exécute le job spécifié en interactif (*foreground*)
%*job*	Exécute le job spécifié en interactif (*foreground*)
kill -9 %*job*	Tue définitivement le job spécifié (équivalent à ^\)
stop %*job*	Stoppe le job spécifié (équivalent à `^Z`). Il peut être repris par `bg` ou `fg`.

Historique des commandes

- Le shell conserve un historique de *n* commandes en définissant la variable de shell:
`set history` *n*
- L'historique est listé par la commande:
history `[-hr] [n]`
`r`: ordre inverse, `h`: sans numéros, `n`: affiche les n dernières commandes.
- L'historique peut être utilisé pour répéter et modifier des commandes:
  - ▷ Désignation de commandes:

!!	désigne la commande précédente
!n	désigne la commande numéro n
!-n	désigne la $n^{ème}$ commande précédente
!*prefix*	désigne la commande la plus récente commençant par *prefix*
!?*chaine*	désigne la commande la plus récente contenant *chaine*
^ch1^ch2	Re-exécute la dernière commande en substituant ch1 par ch2

  - ▷ Désignation de mots:
Des commandes de type `historique:selecteur` permettent de sélectionner des mots dans des commandes de l'historique. Les sélecteurs sont:

0	Le premier mot de la commande spécifiée
n	Le $n^{ème}$ argument
^	Le premier argument
$	Le dernier argument
n1 − n2	Les mots *n1* à *n2*
*****	Tous les arguments
p	Liste la commande désignée sans l'exécuter

  - ▷ Modificateurs de mots:
Des commandes de type `designeur-de-mot:modifieur` permettent d'extraire des parties de mots considérés comme des noms de fichiers (structure `/d1/d2/f.e`). Les modifieurs sont:

h	Nom du répertoire (⇒ `/d1/d2`)
t	Nom du fichier, sans le répertoire (⇒ `f.e`)
r	Racine du nom, sans l'extension (⇒ `/d1/d2/f`)
e	Extension, sans la racine (⇒ `e`)
s/l/r	Substitue *l* par *r*

Alias

- Un alias est une redéfinition de nom de commande UNIX.
- Lors de l'interprétation d'une commande, le c-shell regarde si le premier mot de la commande a été défini comme alias.
- Un alias est défini par la commande:
  alias `nom` *definition*
  ou (selon le shell)
  alias `nom`=*definition* (sans blancs autour du signe égal)
- Un alias peut être supprimé par la commande: **unalias** `nom`
- Tous les alias existant sont listés par la commande `alias` sans paramètres.
- La définition peut utiliser la syntaxe de substitution des commandes d'historique appliquée à la commande courante: `\!:i` représente le $i^{ème}$ argument et `\!*` l'ensemble des arguments
- La définition d'alias peut être court-circuitée en faisant précéder le nom de la commande d'un *backslash* ("\"). La vraie commande sera alors appelée.
- La substitution d'alias ne marche que s'il est le **premier mot** de la commande.
  Ainsi, dans "`if (cond) alias ...`", `alias` ne sera pas reconnu. Il en est de même avec les commandes `time` et `nice`.
- Quelques exemples utiles:

```
alias h history
alias ls ls -lF
alias locate 'who | fgrep \!^'
alias cd 'cd \!:1; set prompt = "$cwd% "'
alias rm '/bin/mv \!*  /.RM'
```

Scripts

Un **script unix** est un fichier de commandes Unix, utilisant divers mécanismes supplémentaires tels que des variables et des instructions de contrôle.

Leur description au chapître sur "*L'écriture des scripts*" page 48 complète les informations sur les shells.

La manipulation de fichiers

- De nombreux utilitaires permettent la manipulation de fichiers organisés en lignes de texte ASCII. Plusieurs d'entre eux permettent l'accès à des champs à l'intérieur des lignes.
- Les fonctionnalités offertes permettent de réaliser jusqu'à de petites bases de données, regroupant plusieurs fichiers. Et sans avoir à écrire le moindre programme en C.
- Les champs dans les lignes d'un fichier sont définis au moyen d'un caractère séparateur de champ. De nombreux traitements peuvent être facilités en choisissant comme séparateur un caractère différent du blanc ou de la tabulation, et en ayant toujours le même nombre de champs dans toutes les lignes d'un fichier.

Recherche de chaînes

- **fgrep**, **grep**, **egrep**: Recherche de chaînes de caractères (`fgrep`) et d'expressions régulières (`grep` et `egrep`) dans un fichier. `GREP` = **G**lobal **R**egular **E**xpression **P**rint (voir le paragraphe suivant).
  - ▷ Chaque ligne contenant la chaîne ou l'expression spécifiée est affichée en sortie
  - ▷ `egrep` utilise un langage d'expression régulières plus étendu que `grep`
  - ▷ syntaxe: `pgm [options]` "`chaine` *ou* `expr.reg`" `[fichier ...]`
  - ▷ options principales:
    - `-i` Ne tient pas compte des maj/min
    - `-n` Précède la ligne affichée du numéro de ligne
    - `-v` Affiche les lignes qui ne contiennent pas la chaîne ou l'expression régulière
- **look** *prefix file*: Recherche la chaîne *prefix* en début de ligne dans un fichier trié
  La recherche est dichotomique, donc très rapide, même dans un grand fichier.

Expressions régulières

Les expressions régulières définissent des modèles de chaînes de caractères destinés à des recherches de chaînes complexes. Elles utilisent un langage particulier défini par les éléments suivants.

Les utilitaires acceptant des expressions régulières sont: `grep`, `egrep`, `sed`, `awk`, `more`, l'éditeur `vi` et tous les languages de script tels que PERL, TCL, PYTHON, etc... Il existe aussi des fonctions C gérant les expressions régulières (`regexp`). Mais le langage peut être plus ou moins complet selon les implémentations.

- Accessible dans tous les utilitaires:

`^`	début de ligne
`$`	fin de ligne
`.`	n'importe quel caractère unique
`[xyz]`	n'importe lequel des caractères spécifiés – ou `[` doivent être placés à la fin de la liste, `]` au début.
`[a-z]`	n'importe quel caractère unique dans l'intervalle spécifié
`[^a-z]`	n'importe que caractère unique en dehors de l'intervalle spécifié
`a*`	zéro ou plusieurs répétitions du caractère `a`

`(ExprReg)`	définition d'une sous-chaîne accessible ensuite par `\i`
$\backslash i$	réutilisation de la sous-chaîne $n^o i$, dans l'expression elle-même ou dans une chaîne de substitution (`vi` ou `sed`).
$\backslash car$	marque un caractère réservé pour lui ôter son rôle particulier
`&`	Lorsqu'une expression régulière est utilisée dans une commande de substitution, le caractère & peut être utilisé pour représenter la chaîne identifiée par l'expression régulière

- Accessible seulement dans **sed**:

`a{m}`	exactement m occurences du caractère `a`
`a{m,}`	au moins m occurences du caractère `a`
`a{m,n}`	entre m et n occurences du caractère `a`

- Accessible seulement dans **awk** et **egrep**:

`|`	OU logique entre plusieurs expressions
`a+`	une ou plusieurs répétitions du caractère `a`
`a?`	zéro ou une occurence du caractère `a`

- Accessible seulement dans **vi** et **grep**:

`\<`	début de mot
`\>`	fin de mot

Le traitement des expressions régulières dans **vi** est inhibé par la commande: "*:set nomagic*".

Exemples:

`^A.*Z$`	Ligne de n'importe quelle longueur commençant par A et se terminant par Z.
`\<[A-Z][A-Z0-9]*\>`	Tout mot contenant des caractères alphanumériques mais commençant par une lettre (dans vi)
`"^.  "`	Toute ligne ayant un blanc en deuxième caractère
`^\(..*\)\1$`	Toute ligne contenant deux fois la même chaîne (dans sed et grep)
`alpha|delta`	Reconnait la chaîne `"alpha"` ou `"delta"` (`awk` ou `egrep`)

Extraction de données

- **head** $-nb$: Affiche les nb premières lignes d'un fichier

- **tail** `[-fr]` $\pm nb$: Affiche les dernières lignes d'un fichier
  Exemples:

`tail -nb fich`	Affiche les nb dernières lignes du fichier
`tail +nb fich`	Affiche depuis la ligne nb jusqu'à la fin du fichier
`tail -r fich`	Affiche un fichier en inversant l'ordre des lignes
`tail -f fich`	Continue à lire après la fin du fichier. Utile pour un fichier en cours d'écriture (Doit être interrompu par ^C).

- **cut** `[-d]`sep `{-c|-f}`$liste$: Extrait des colonnes d'un fichier
  Cela peut être des colonnes de caractères (option `-c`) ou des colonnes de champs (option`-f`), avec un séparateur défini par l'option `-d`sep. La liste de colonnes ou de champs est de la forme $n1-n2, n3-n4$...

Fusion de fichiers

- **paste** *fich1 fich2* ... : Fusion de fichiers ligne à ligne
  - ▷ Les fichiers doivent avoir le même nombre de lignes.
  - ▷ Le fichier standard d'entrée peut être utilisé en le désignant par "–".
  - ▷ Utilisation particulière: `paste - - - ...`
    Regroupe n lignes du fichier standard entrée en une seule.
- **join**: Jointure de fichiers au sens relationnel
  La jointure est faite en définissant dans chaque fichier un champ comme clé primaire.

Comparaison de fichiers et de répertoires

- **diff**: Compare deux fichiers et affiche les différences
  `diff` a des difficultés à fonctionner au-delà de quelques milliers de lignes. Il faut alors utiliser `bdiff` ou `diff -h` selon les systèmes.
- **comm** `[ -[1][2][3] ] fich1 fich2`: Comparaison de fichiers triés
  Sortie possible de 3 colonnes (selon les options):
  1: supprime les lignes présentes seulement dans le fichier 1
  2: supprime les lignes présentes seulement dans le fichier 2
  3: supprime les lignes présentes dans les deux fichiers
- **cmp**: Comparaison binaire de fichiers. La comparaison s'arrête lorsque une différence est trouvée.
- **dircmp**: Différence de répertoires (voir aussi `diff`).
- **diff3**: Comparaison de 3 fichiers

Tri de fichiers

- **sort**: Tri de fichier
  - ▷ Appel: `sort [options] [cle ...] [fich ...]`
  - ▷ `sort` peut trier sur les lignes entières ou sur n'importe quelle clé définie dans un champ.
  - ▷ Un champ est défini comme une suite de caractères dans une ligne séparé du prochain par un caractère séparateur (blanc ou tabulation par défaut).
  - ▷ Chaque clé est de la forme: ±`w.c` ou:
    - `+` indique le début d'un champ à trier
    - `-` indique la fin du champ à trier
    - `w` indique le nombre de champs à sauter dans la ligne. ...ou le numéro du champ, si on commence le numérotage à zéro.
    - `c` indique le nombre de caractères à sauter dans le champ.

Exemple (le caractère séparateur est le ":"):

```
 champ0 :champ1:champ2: champ3
------------------------------
t999abcd:12.345: 12345:09.10.93
------------------------------
    ++++                  ++
    crit1                crit2
```

Le tri sur les deux critères désignés se fait par: `sort -t:  +0.4 -1 +3.3n -3.5` *fich*

▷ Les options suivantes peuvent s'appliquer à l'ensemble des clés, ou à certaines en particulier:

`f`	Ignore les différences maj/min
`n`	Traite le champ comme un nombre, pas comme une chaîne de caractères
`r`	Tri en ordre inverse

▷ Les options suivantes sont globales:

`-t`car	Utilise car comme caractère séparateur de champ
`-u`	Ne sort qu'une ligne si plusieurs ont les mêmes critères de tri. Voir aussi la commande `uniq`
`-o fich`	La sortie est mise dans le fichier spécifié (au lieu de *stdout*)

- **uniq** `[-duc] [`*f-lect* `[`*f-ecr*`] ]`: Gère les répétitions dans un fichier trié

`-d`	Affiche seulement une copie de chaque ligne répétée
`-u`	Affiche seulement les lignes non répétées
`-c`	Précède chaque sortie du nombre d'occurences de la ligne en entrée

Modification de fichiers

- **tr**`[-cds] liste1 liste2 <` *f-entree* `>` *f-sortie* : Remplace des caractères dans un fichier

  ▷ `tr` n'accepte pas de noms de fichiers comme paramètres. Ils doivent être définis par des redirections ou des *pipes*.

  ▷ `tr` remplace chaque caractère de la liste 1 par le caractère correspondant de la liste 2.

  ▷ Une liste de caractères peut être constituée de plusieurs caractères accolés (`abc`), de listes spécifiées par des limites (`A-Z,a-z0-9`), ou de nombres octaux (`'\012'`).

  ▷ Options:

`-d`	Supprime tous les caractères de la liste1 trouvés dans le fichier
`-s`	Réduit toute suite de caractères en sortie, qui sont dans la `liste2`, à un seul caractère
`-c`	Utilise comme *liste1* tous les caractères ASCII qui n'y figurent pas

  ▷ Exemples:

`tr A-Z a-z < fich1 > fich2`	Convertit toutes les majuscules de fich1 en minuscules dans fich2
`tr -cs A-Za-z '\012'`	Traduit un fichier quelconque en une liste de mots alphabétiques, à raison de 1 mot par ligne

- **sed**: Editeur batch, c'est à dire non interactif.
  A l'aide d'un ensemble de commandes d'éditeur, cet utilitaire est un moyen très puissant de modifier un fichier d'une façon connue à l'avance.

  ▷ Appel: `sed commande [fichier ...]`

  ▷ Les commandes sont de la forme:
  `[adresse1 [,adresse2] ] fonction [parametres]`

  ▷ Principales options:

`-n`	Pas de sortie après modification Ex: Affiche les lignes contenant une expression régulière (comme grep): `sed -n '/expr-reg/p' fich`
`-e`	L'option est suivie d'une commande. Elle n'est nécessaire que si l'on veut écrire plusieurs commandes Ex: `sed -e 's/A/a/g' -e 's/Z/z/g' fich`
`-f` *fcmd*	Les commandes de modifications sont dans le fichier *fcmd*. Cette option est très utile si la commande contient des caractères d'expressions régulières qui interfèrent avec les caractères spéciaux utilisés par le shell ($, * par exemple)

▷ Dans la chaîne de substitution, le caractère & peut être utilisé pour représenter l'expression régulière
Ex: `s/abc/&&/` remplacera "`abc`" par "`abcabc`".

▷ De même, `\i` represente la $\texttt{i}^{ème}$ expression régulière mise entre parenthèses dans la première partie.

▷ Quelques exemples classiques:

`sed 's/Unix/UNIX/g ; s/dea/DEA/g' fich`	Remplace toutes les occurences de ces mots
`sed 's/$/\n/' fich`	Double les interlignes
`sed '/^$/d' fich`	Supprime toutes les lignes vides
`sed '/^[ \t]*$/d' fich`	Supprime toutes les lignes vides ou ne contenant que des blancs et des tabulations
`sed 'y/abc/ABC/' fich`	identique à `tr 'abc' 'ABC' fich`

- **awk**: Manipulation programmable de fichiers

▷ Le nom `awk` vient des initiales des trois auteurs: Alfred **A**ho, Peter **W**einberger et Brian **K**ernighan.

▷ `awk` est un utilitaire qui permet des traitements complexes sur les lignes d'un fichier.

▷ Appel: `awk [-F`sep`] programme [fichier ...]`
Le programme peut aussi être écrit dans un fichier et appelé par l'option: `-f fichier-programme`

▷ Le programme décrivant les traitements utilise un langage interprété dont les instructions ont la structure suivante:

CONDITION { **ACTION** }

`awk` lit le fichier de données ligne par ligne. Pour chaque ligne, il teste la partie CONDITION de chaque instruction du programme. Si la CONDITION est vraie pour la ligne courante, alors la partie ACTION est exécutée pour cette ligne.
Une instruction peut être écrite sur plusieurs lignes en indiquant qu'il y a une suite par un "\" comme dernier caractère d'une ligne.

▷ Dans une ligne, les différents champs sont accessibles par des variables $1, $2, ...
$0 représente la ligne complète.
Le séparateur de champ (blancs par défaut) peut être redéfini (option `-F` ou variable `FS`).

▷ La CONDITION peut être une expression logique ou une expression régulière entourée de "/".
Il y a aussi deux conditions particulières: `BEGIN` exécuté avant le début du fichier, et `END` exécuté après la fin du fichier. Exemples: "`$1 == "alpha"`" "`/ A.*/`"

▷ Les expressions utilisent les opérateurs suivants:

⋆ arithmétiques: `+ - * / % ^ ++ --`

⋆ logiques: `&& || ! > < >= <= == !=`

⋆ de reconnaissance d'expressions régulières: `~`: contient..., `!~`: ne contient pas... l'expression régulière.

⋆ de concaténation de chaînes: l'espace (opérateur implicite).

▷ L'ACTION est constituée d'instructions proches du C, comportant les éléments suivants:

⋆ Séparateur d'instructions: point-virgule ou saut de ligne

⋆ Plusieurs instructions peuvent être regroupées en une seule en les entourant d'accolades {`instr1 ; instr2 ; ...`}.

⋆ Assignation: var = expr
On peut aussi utiliser les opérateurs suivants: `+= -= *= /= %=` qui ont la même signification qu'en C (`var += val` ⇔ `var = var + val`).

⋆ Variables:

· **Les variables de champs**: définissant les champs de la ligne courante du fichier:
$0 pour la ligne complète. $1,$2... pour les champs successifs.

· **Variables utilisateurs**: définies par des caractères alphanumériques.
La valeur d'une variable peut être une chaîne de caractères ou un nombre. Elle peut être forcée dans un type différent en lui associant une opération vide du type que l'on désire:
`num = "123" + 0` fera considérer `num` comme un nombre
`str = num ""` forcera `str` à être une chaîne de caractères

· **Variables pré-définies**: Ces variables sont disponibles dans tout programme awk:

FS	Séparateur de champs (aussi définissable par l'option -Fsep)
RS	séparateur d'enregistrements en entrée (défaut: '\n')
NF	Nombre de champs de l'enregistrement courant
FNR	Nombre d'enregistrements lus dans le fichier courant
NR	Nombre total d'enregistrements lus depuis le début de l'exécution
FILENAME	Nom du fichier courant

· **Tableaux de variables**:
Awk gère des tableaux de variables dont les indices peuvent être indifféremment des nombres ou des chaînes de caractères. On peut écrire:
`tab[3] = "abc"` ou
`tab["abc"] = 3`

⋆ Fonctions internes:

`length(s)`	nombre de caractères dans la chaîne (sans arguments: longueur de la ligne)
`index(s,t)`	position de la chaîne t dans la chaîne s
`substr(s,p[,n])`	extrait de la chaîne s la sous-chaîne commençant en position p et ayant n caractères de long. Par défaut, elle sera prise jusqu'à la fin.
`sprintf(fmt,expr,...)`	retourne la chaîne construite à partir du format et des arguments. Le format utilise la même écriture que les formats en C.
`sqrt(n)`	racine carrée de n

Des versions étendues de `awk` (`gawk`, `nawk`) comportent en plus des fonctions de reconnaissance et de substitution d'expression rgulières (`match`, `sub`, `gsub`, ...), ainsi que la possibilité de définir ses propres fonctions.

⋆ Sorties:

`print expr,...`	imprime les expression, sans formattage particulier
`printf fmt,expr,...`	imprime les expression, selon le format spécifié. Ceux-ci utilisent la même écriture que les formats en C.

⋆ Instructions de contrôle:

· `if (condition) instr1 [else instr2]`

· `for (initialisation ; test de sortie ; increm) instruction`

· `for (indice in tableau) instruction`

· `while (condition) instruction`

· `break`: permet de sortir de la boucle `for` ou `while` la plus récente.

· `continue`: va directement à la partie incrémentation, à la fin de la boucle courante.

· `next`: Saute toutes les instructions restantes et recommence tout le programme avec la ligne suivante du fichier d'entrée.

· `exit`: sortie définitive de `awk`.

▷ Exemple:
Soit un fichier `exemple.dat` contenant des enregistrements de structure: `nom:i1:i2:d`

```
aa:12:23:123.45
bb:6:2:7.9
aa:142:1:1081.56
aa:7:15:634.10
bb:9:20:811.47
```

Le programme `exemple.awk` suivant fait:

- ⋆ Une édition formattée du fichier de données dans un fichier de même nom suffixé "`.new`".
- ⋆ Un cumul de la colonne "d" par `nom`
- ⋆ Une recherche du "d" maximum
- ⋆ Un affichage à la fin du fichier des cumuls par `nom` et du maximum

```
BEGIN \
{                       # le \ est necessaire pour signifier que
                        # l'action qui suit est liee a cette condition.
   FS = ":"             # defini le separateur
   max = 0.0            # initialise le maximum.
                        # peut etre omis, car toute variable est
                        # initialise a 0 ou a une chaine vide ("")
                        # selon son usage ulterieur
}

#calcule un nouveau maximum si necessaire
$4 > max { max = $4 ; maxnom = $1 }

# pour toutes les lignes, fait une edition formattee de l'entree
# dans un fichier de meme nom avec le suffixe '.new'
{
   printf "%-4s: %4d : %4d : %8.2f\n",$1,$2,$3,$4 > FILENAME ".new"
   val[$1] += $4        # calcule le cumul de 'd' pour le nom courant
}

# A la fin du fichier, affiche les cumuls de la colonne 4 par nom
# ainsi que le maximum
END \
{
   for (nom in val)
      printf "%-4s : %10.2f\n",nom,val[nom]
   printf "Maximum: %-4s ==> %8.2f\n",maxnom,max
}
```

Le fichier résultat `exemple.dat.new` sera:

```
aa  :   12 :   23 :   123.45
bb  :    6 :    2 :     7.90
aa  :  142 :    1 :  1081.56
aa  :    7 :   15 :   634.10
bb  :    9 :   20 :   811.47
```

Et l'affichage à l'écran:

```
aa  :    1839.11
bb  :     819.37
Maximum: aa ==>  1081.56
```

- **split** $-nblig$: Décompose un fichier en fichiers ayant tous le même nombre de lignes.
- **fold** $-lg$: Découpe les lignes d'un fichier en lignes de longueur fixe
- **expand**: Remplace les tabulations d'un fichier par des blancs
- **unexpand**: Remplace les caractères blancs d'un fichier par des tabulations.

Le langage Perl

Le langage Perl est un langage de programmation complet, de type semi-interprété, offrant de façon intégrée les fonctionnalités de gestion de données contenues dans les commandes Unix. Une gestion très complète des expressions régulières offre des écritures très concises pour de petites applications. Le langage dispose aussi de nombreuses interfaces avec les fonctions systèmes.

Ce langage est l'un des plus utilisés pour l'écriture de scripts Web.

Documentation:

- *Programming Perl*. Larry Wall, Tom Christiansen and R. L. Schwartz. O'Reilly & Associates.
- *Advanced PERL programming*. Sriram Srinivasan. O'Reilly & Associates.
- *Perl Résumé*. M. Wenger. Observatoire de Strasbourg
- `http://www.perl.com/`

L'écriture de scripts (C-shell)

Introduction

- un shell est un interprète d'un langage de programmation particulier
- un *shell-script* est équivalent à un programme exécutable. C'est un fichier qui contient des commandes UNIX mélées à des variables de shell et des instructions de contrôle, qui sont en général des commandes internes au shell. Le fichier doit être rendu exécutable par la commande `chmod +x script`.
- Exemple:
  Ce script permet de supprimer une liste des fichiers en demandant confirmation pour chacun d'entre eux.

```
#!/bin/csh
# rmd liste-de-fichier
# remove avec dialogue
#
if ("$1" == "") then                  # s'il n'y a aucun argument
   echo "Usage: $0 liste-de-fichier"  # le script affiche un bref mode d'emploi
   exit 1                             # et se termine
endif
set FLIST = ($*)                      # FLIST contient la liste de ts les param
foreach ff ($FLIST)                   # pour chaque element de la liste
   if (! -f $ff) then                 # teste si l'element est un fichier simple
      echo "$ff n'est pas un fichier"
      continue                        # passe au suivant sinon
   endif
   ls -l $ff                          # affiche les caracteristiques du fichier
   question:                          # label. destination d'un goto
   echo -n "Voulez vous supprimer $ff (oui/non) ?"  # affiche la question
   set rep = $<                       # attend et recupere la reponse
   switch($rep)                       # aiguillage selon la reponse
      case [oO]*:                     # commence par 'o' ou 'O'
         rm $ff                       # supprime le fichier
         echo "Le fichier $ff a ete supprime"
         breaksw
      case [nN]*:                     # commence par 'n' ou 'N'
         echo "Le fichier $ff n'est pas supprime"
         breaksw
      default:                        # reponse non reconnue par les 'case'
         echo "Reponse incorrecte"
         goto question                # va au label question
   endsw                              # fin du switch($rep)
end                                   # fin du foreach ff ...
exit                                  # sortie du script avec un statut 0
```

Exécution d'un shell

- Deux façons d'exécuter un shell:
  - ▷ Par **interprétation directe**: en appelant le shell et en lui fournissant le script en paramètre: `csh [options]` *script parametres* ...
  - ▷ Par **interprétation indirecte**:
    Il faut d'abors rendre le script exécutable: `chmod +x script`
    Puis, il peut être appelé comme n'importe quelle autre commande: `script` *parametres*
- Si une commande vient d'être crée dans un répertoire du *path* autre que le répertoire courant, il faut exécuter la commande **rehash** pour rendre la commande accessible. **rehash** recrée une table, interne au shell, des commandes accessibles.
- Un script exécutable apparait suivi d'un "*", avec la commande `ls -F` (comme un exécutable).
- Sélection du shell pour interpréter le script:
  - ▷ Par défaut, le Bourne-shell (`sh`) est utilisé pour exécuter un script.
  - ▷ Si le premier caractère de la première ligne est un `#`, alors le shell par défaut (*login shell*) est utilisé.
  - ▷ L'utilisateur peut forcer n'importe quelle commande à interpréter son script, en écrivant cette commande en première ligne du script, précédée de `#!/...` (L'interprête doit être donné avec son chemin absolu):

    ```
    #!/chemin/programme-interprete
    ... lignes contenant un programme et/ou
    ... des donnees
    ...
    ```

  - ▷ Pour se prémunir contre toute mauvaise interprétation, il est recommandé de toujours préciser le shell que l'on souhaite utiliser: `#!/bin/csh` ou `#!/bin/sh` par exemple, sur la première ligne d'un script.

Variables d'environnement

- Elles sont associées à un process et disparaissent avec lui.
- Elles sont héritées par tout process fils.
  Ainsi, un programme C pourra y accéder par la fonction `getenv()`.
- Elles sont traitées par les commandes suivantes:

setenv `var` *val*	Affecte une valeur à une variable d'environnement
unsetenv `var`	Supprime une variable d'environnement
printenv	Liste les variables d'environnement

- L'utilisation et la substitution des variables d'environnement est le même que pour les variables locales.

- Certaines variables d'environnement sont définies au démarrage d'une session:

`HOME`	Home répertoire de l'utilisateur (variable d'environnement)
`USER`	Nom de l'utilisateur (*username*)
`PATH`	Liste des répertoires constituant les chemins d'accès aux programmes
`SHELL`	Shell courant. Utilisé pour l'exécution de sous-commandes par certains utilitaires (vi,...)
`TERM`	Type de terminal utilisé
`MAIL`	Fichier de mail primaire (cf p 62)

- Les variables d'environnement sont souvent utilisées pour le paramétrage de logiciels.
  Exemple: les variables d'environnement utilisées par TEX et LATEX:

```
IDL_DIR=/usr/local/rsi/idl_5.2
IDL_PATH=+/usr2/rsi/idl_5.2/lib:+/usr/local/idl_aulib:/atmos_work/Bin
PGPLOT_DIR=/usr/local/pgplot
```

Les variables locales du shell

- Elles sont associées à un shell et ne sont *visibles* que par ce shell. Elles ne sont pas transmissibles.
- Une variable locale peut porter le même nom qu'une variable d'environnement. C'est la variable locale qui a précédence sur la variable d'environnement lors d'une substitution.
- Une variable contient toujours une chaîne de caractères.
  Celle-ci peut être considérée comme un nombre si tous les caractères sont numériques.
  Une variable peut être considérée comme un tableau de sous-chaînes.
- Elles sont traitées par les commandes suivantes:

set `var`	Défini une variable sans lui affecter de valeur
set `var` = *val*	Affecte une valeur à une variable locale Pour un tableau, les éléments doivent être entre parenthèses, séparés par des blancs
@ `var` = *expr*	Semblable à `set`, mais permet l'utilisation d'opérateurs de type C pour calculer des expressions arithmétiques
set `var[N]` = *val*	Affecte le N^{ème} mot d'une variable tableau. Le tableau doit avoir au moins N valeurs
unset `var`	Supprime une variable locale
set	Liste toutes les variables locales

Substitution de variables

- Une variable est remplacée par sa valeur dans un shell au moyen des syntaxes suivantes:

`$var`	donne la valeur d'une variable
`${var}`	pour séparer une variable des caractères suivants
`$var[i]`	donne le $i^{ème}$ mot d'une variable tableau
`$var[i-j]`	donne les mots i à j d'une variable tableau
`$var[i-]`	donne les mots du $i^{ème}$ au dernier d'une variable tableau
`$var[i-j]`	donne les mots du premier au $i^{ème}$ d'une variable tabelau

`$var:mod`	modificateur de variable (comme pour l'historique):
	utilisable pour des variables représentant des noms de fichier.
	Ex: `/d1/d2/fich.ext`
	les modificateurs possibles sont:
	`h` : head : chemin (jusqu'au dernier "/"): ⇒ `/d1/d2`
	`t` : tail : fichier sans le chemin ⇒ `fich.ext`
	`r` : root : racine du nom de fichier ⇒ `/d1/d2/fich`
	`e` : extension : extension du fichier ⇒ `ext`
	`gh,gr,gt,ge` : s'applique globalement à tous les éléments d'un tableau

- Il existe des substitutions retournant autre chose que la valeur d'une variable:

`$#var`	donne le nombre de mots d'une variable tableau
`$?var`	Vaut 1 (vrai) si la variable a été définie, 0 (faux) sinon. Utilisé dans des instructions de test

- La substitution des variables n'est pas faite si la variable est entourée de simples quotes.

Variables prédéfinies

- Variables associées aux arguments du script:
  Un script peut être appelé avec des arguments sur la ligne de commande. Un argument consiste en un mot, séparé du prochain par des blancs. Les variables suivantes permettent d'utiliser ces arguments dans le script:

`$argv`	contient la liste complète des arguments
`$#argv`	est le nombre d'arguments dans `argv`
`$argv[i]`	est la valeur du $i^{ème}$ argument
`$i`	est équivalent à `$argv[i]`, avec i $\leq$ 9
`$*`	représente l'ensemble des arguments

- Les arguments proprement dit sont numérotés à partir de 1. `$0` représente le nom du script (`$argv[0]` n'est pas utilisable).

- La commande "`shift` *variable*" permet de décaler les éléments dans une variable de type tableau, en éliminant le premier élément de la liste. Par défaut, c'est `argv`, la liste des paramètres de la commande, qui est décalé.

- Autres variables:

`$0`	Le nom du script
`$$`	Le numéro de process du shell. Utile pour créer dans un script des fichiers temporaires ayant un nom garanti unique sur la machine.

`$<`	Lit une ligne en *standard input* et retourne la chaîne tapée. Utile pour rendre un script interactif avec question/réponses.
	Exemple:
	
	`debut:`
	`  ...`
	`echo -n "Voulez-vous continuer (o/n) ?"`
	`set rep = $<`
	`if ("$rep == o") goto debut`
	`  ...`

`$status`	Le statut de retour de la dernière commande exécutée dans le script

Substitution de commandes

- Une commande Unix placée entre *backquotes* (“‘”) est exécutée par un sous-shell.
- Le résultat de son exécution, sa sortie standard, est décomposée en mots, tous sur la même ligne. Les sauts de ligne sont remplacés par des espaces.
- Ce résultat est substitué à la commande.
- Exemples:

```
set fichiers = ‘ls‘
set dir_courant = ‘pwd‘
set DATE = ‘date +’%y.%m.%d’‘
```

Marquages particuliers: backslash et apostrophes

- **backslash**: “\”
  S'applique au caractère qui suit, en lui supprimant son rôle éventuellement particulier dans l'interprétation du shell. Ex: Comparer `echo *` et `echo \*`.
- **Apostrophes simples**: `('...')`
  - ▷ Utilisable pour toute chaîne de caractères
  - ▷ Seule la substitution d'historique (cf p 37) est appliquée à la chaîne.
- **Apostrophes doubles** `("...")`
  - ▷ Utilisable pour toute chaîne de caractères comme les simples apostrophes.
  - ▷ Les substitutions de variables, de noms de fichiers et de commandes sont effectuées.
  - ▷ Dans des substitutions de commandes, seuls les sauts de ligne sont considérés comme des séparateurs de mots.
  - ▷ Exemple:

```
% set date_array = ‘date‘
% set date_string = "‘date‘"
% echo date_array \| date_string
Thu Sep 23 10:57:01 MET DST 1993 | Thu Sep 23 10:57:02 MET DST 1993
% echo "#date_array | #date_string"
6 | 1
```

    `date_array` est un tableau de mots indépendants. A cause des doubles apostrophes (") utilisées à sa création, `date_string` est considéré comme une simple chaîne de caractères.

Commentaires

Tout ce qui est écrit derrière un # dans une ligne d'un script est considéré comme commentaire.

Expressions et opérateurs

- Toutes les valeurs manipulées dans un script sont des chaînes de caractères

- Des chaînes de chiffres peuvent représenter des valeurs numériques

- Tous les opérateurs C sont disponibles:
  `+ - / * % < > <= >= == != && || !`
  Il y a en plus deux opérateurs de comparaison avec des expr. reg.:
  `=~ !~` (contient, ne contient pas l'expression régulière).

- Test spéciaux pour fichiers:

`-r fich`	vrai si le fichier est lisible par l'utilisateur
`-w fich`	vrai si le fichier peut être écrit par l'utilisateur
`-x fich`	vrai si le fichier est exécutable par l'utilisateur
`-e fich`	vrai si le fichier existe (quel que soit le type)
`-o fich`	vrai si l'utilisateur est le propriétaire du fichier
`-z fich`	vrai si le fichier est vide
`-f fich`	vrai si le fichier est un simple fichier (pas un dir, un lien, ...)
`-d fich`	vrai si le fichier est un répertoire

- Test spécial utilisant des commandes:
  `{ command }` : vrai si la commande se termine avec un status zéro

- Test de l'existence d'une variable:
  `$?variable`

Instructions de contrôle

- **instruction If-then**
  - ▷ Forme longue:

```
if (expr) then
    ...
else if (expr2) then    # les else if et
    ...                 # else sont
else                    # facultatifs !
    ...
endif
```

  - ▷ Forme courte:

```
if (expr) simple_commande
```

    La commande doit obligatoirement tenir sur une seule ligne.

- **Boucle While**
  Le programme boucle tant que l'expression est vraie.

```
while (expression)
    ...
end
```

- **Boucle Foreach**
  `foreach` exécute ses instructions une fois pour chaque valeur de la liste de mots. A la $i^{ème}$ itération, `var` a la valeur du $i^{ème}$ mot du tableau *liste-de-mots*.

```
foreach var (liste-de-mots)
    ...
end
```

- **Instructions associées à ces boucles**
  - ▷ **continue**: va au prochain pas de la boucle courante.
  - ▷ **break**: interrompt l'exécution de la boucle courante.

- **Instruction Switch**
  Branchement multiple basé sur la valeur d'une chaîne de caractères. Les sélecteurs "`case chaine:`" peuvent contenir des expressions de désignation de fichiers, avec *wildcards*.
  Les instructions `breaksw` sont indispensables, sinon le programme continue dans la prochaine branche.

```
switch (string)
    case nom1:
        ...
    breaksw
    case nom2:
        ...
    breaksw
    default:
        ...
    breaksw
endsw
```

- **Commande "exit": sortie du shell**

La commande `exit` permet de sortir d'un shell en retournant un statut qui peut ensuite être testé.

```
exit [ (expr) ]
```

- **Instruction goto et label**

```
label:
    ...
goto label
```

- **Instruction repeat**
  Répète N fois la commande spécifiée.

```
repeat N cmd
```

Commandes diverses utiles dans un script

`echo`	Recopie les arguments sur la sortie standard L'option `-n` supprime le `\n` à la fin de la ligne.
`clear`	Efface l'écran
`date`	Affiche la date et l'heure. format particulier possible avec la syntaxe: `date` *+format* ou *format* est un format de type C avec les descripteurs suivants (précédés de %): d Jour du mois en 2 chiffres (01 – 31) m Mois sur deux chiffres (01 – 12) h Mois en trois lettres ('Jan' – 'Dec') y Deux derniers chiffres de l'année Y Année avec quatre chiffres j Jour de l'annee en trois chiffres (000 – 366) w Jour de la semaine en 1 chiffre (0:dimanche,. . . ,6:samedi) a Jour de la semaine en trois lettres ('Sun' – 'Sat') Z Zone de temps (ici: MET) H Heure en deux chiffres (00 - 23) M Minutes en deux chiffres (00 – 59) S Secondes en deux chiffres (00 – 59) D Date sous la forme `mm/jj/aa` T Heure sous la forme `hh:mm:ss`

La création de programmes

Le langage C

- **Le source** est le *texte* du programme, écrit au moyen d'un éditeur de texte.
- **Le module objet** est le résultat de la **compilation** du source. C'est la traduction du programme source en **langage machine** au moyen d'un **compilateur**. Un langage machine est lié à une famille de processeurs (Sparc, Alpha, Intel,...).
- **Le programme exécutable** est obtenue par le regroupement des modules objets constituant le programme et de ceux existant dans les **librairies** nécessaires au moyen de **l'éditeur de lien**.
- Par convention, utilisée par un certain nombre d'utilitaires, les fichiers sont reconnus par l'utilisation d'extensions standards:

.c	Source C	**.o**	Module objet	**.l**	Sources lex
.C .cc .cpp .cxx	Source C++	**lib***xxx***.a**	Librairie *xxx* statique	**.y**	Sources yacc
.h	Headers C	**lib***xxx***.so**	Librairie *xxx* partageable		

- Une compilation – simple – se fait par la commande:
  `cc -c monprog.c -Iinclude_dir`
- Une édition de lien entre des modules objets et des librairies se fait par la commande:
  `cc monprog.o ssp.o ... -Llibr_dir -lxxx -o monprog`
  Parmi les modules objets, il doit y en avoir un et un seul qui contienne une fonction **main**().
- La compilation et l'édition de lien peuvent se faire en une commande:
  `cc monprog.c ssp.o ... -Iinclude_dir -Llibr_dir -lxxx -o monprog`
  La forme la plus simple est: "`cc monprog.c`" qui produit un exécutable appelé "`a.out`".
- Le programme est ensuite exécuté comme toute commande UNIX:
  `monprog` [*parametres ...*]
- Les librairies partageables (`.so`) doivent être dans des répertoires définis dans la variable d'environnement `LD_LIBRARY_PATH`.
- Les fichiers standards UNIX sont spécifiés en C par les noms `stdin`, `stdout` et `stderr`, définis dans *stdio.h*.
- Les arguments de la ligne de commande sont accessibles en C par le mécanisme suivant:

```
main(argc,argv)
        int argc ;    /* nombre d'arguments */
        char **argv ; /* pointeur sur un tableau des adresses des arguments
                       *    argv[0] = nom du programme
                       *    argv[i] = argument numero i
                       */
{
...
        p3 = argv[3] ;      /* 3eme argument */
        c42 = argv[4][2] ;  /* 3eme caractere du 4eme argument */
        c20 = *argv[2] ;    /* 1er caractere du 2eme argument */
...
```

Utilisation de librairies de modules objets

- Une librairie de modules objets est un fichier contenant un ensemble de sous-programmes déjà compilés. L'éditeur de liens peut en extraire ceux dont il a besoin pour construire un programme exécutable.

- Une édition de liens peut être:

  ▷ **statique:** les modules et fonctions nécessaires seront physiquement inclus dans l'exécutable.

  ▷ **dynamique:** seule une référence au module est incluse dans l'exécutable. Le chargement physique du module se fera à l'exécution.

- Il est recommandé de nommer une librairie en respectant la syntaxe: **lib***xxx***.a** ou **lib***xxx***.so**. Une librairie statique sera incluse explicitement lors d'une édition de lien par les options `-L`*lib_dir* `-l`*xxx*.
  Il est aussi possible de désigner une librairie à utiliser par son nom complet, hors conventions:
  "`cc monprog.c malibrairie ...`".
  Une librairie partageable sera accédée si son répertoire est dans la liste des répertoires de la variable d'environnement `LD_LIBRARY_PATH`:
  `setenv LD_LIBRARY_PATH ".;/usr/local/lib;/usr/lib"`

- Des modules objets sont écrits ou remplacés dans une librairie par la commande:
  `ar rv libxxx.a obj1.o obj2.o ...`

- La table des points d'entrées dans la librairie doit être créee après chaque modification de la librairie par la commande:
  `ranlib libxxx.a`

- Des modules objets peuvent être supprimés d'une librairie par la commande:
  `ar -dv libxxx.a obj1.o ...`

Les utilitaires d'aide au développement

- **make**: Aide à la construction de programmes exécutables

  ▷ Un fichier **Makefile** contient une description de la construction d'un programme sous la forme de buts et de dépendances.

  ▷ La syntaxe principale du fichier *Makefile* est:

```
but: dependances
<TAB> commande unix
      ...
```

    ⋆ Les **dépendances** sont des buts à réaliser avant le but courant.

    ⋆ Un **but** est **réalisé** lorsque sa date de mise à jour est plus récente que celles de ses dépendances.

    ⋆ Les **commandes** sont celles nécessaires à la mise à jour du but. Elles seront exécutées lorsques toutes les dépendances auront été mises à jour.
    Au moins un caractère de tabulation (<TAB>) est obligatoire devant chaque commande.

  ▷ Le langage des makefile permet:

    ⋆ la définition de variables d'environnement:

```
CC = cc
CFLAGS = -O -Imondir
OBJ=monprog.o,monssp1.o \
    monssp2.o
```

    ⋆ la définition d'actions par défaut basées sur les suffixes:

```
.SUFFIXES: .o .c
.c.o:
<TAB>$(CC) $(CFLAGS) -c $<
```

  ▷ Appel de make:

make	exécute make en réalisant le 1er but défini dans le fichier
make prog	exécute make en réalisant le but spécifié
make clean	*clean* est un but souvent défini pour nettoyer un répertoire des fichiers temporaires (.o, exécutables, etc ...)
make -n	permet de voir quelles commandes seraient exécutées

▷ La commande **touch** *fich* permet de changer la date de dernière mise à jour d'un fichier, forçant ainsi des dépendances à être reexécutées.

▷ Exemple de Makefile
L'exemple suivant permet, dans un répertoire, de développer une librairie de fonctions, avec plusieurs programmes les utilisant. Il peut être utilisé tel quel, en remplaçant les listes de modules objets (`OBJ=..`), les librairies (`LIB=..`) et les programmes (`PGM=..`).

```
#
CC = cc                                 # choix du compilateur
CINCL = -I.                             # includes explicites
CFLAGS = -O                             # options std pour le compilateur C
CLIBS =                                 # librairies utilisateurs

# Liste des modules objets a gerer:
OBJ = simcli.o uifclient.o skclient.o skio.o

# Librairie a creer
LIB = libsimcli.a

# Liste des programmes a maintenir
PGM = clitest clibib cliname clicoord clitime clifct

# Definition de regles par defaut
.SUFFIXES: .o .c
.c.o:
        $(CC) $(CFLAGS) $(CINCL) -c $<


all: $(LIB) $(PGM)                      # but initial et par defaut

$(LIB): $(OBJ)                          # construction de la librairie
        ar sruv $@ $(OBJ)

$(PGM): $$@.o $(LIB)                    # construction des programmes
        cc $@.o $(LIB) $(CLIBS) -o $@

clean:                                  # nettoyage du r\'epertoire
        rm -f $(PGM) $(LIB) *.o core
```

- **SCCS**, **RCS**: gestionnaires de sources, avec maintenance des versions successives.
- **lex** et **yacc**: Analyseur lexical et syntaxique (YACC = Yet Another Compiler Compiler).
- **dbx**: Aide à la mise au point.
  Au moins utile pour récupérer le nom du module et le numéro d'instruction lors d'un plantage (qui se traduit en général par un laconique "`segmentation fault (core dumped)`":

  ▷ compiler le ou les modules avec l'option "-g".

  ▷ après l'exécution qui a planté, faire:

  ```
  dbx monprog core
  (dbx) where
  (dbx) quit
  ```

  On récupère la chaîne des appels des sous-programmes, plus l'endroit du plantage.

  De nombreuses autres commandes permettent une exécution pas à pas d'un programme, d'implanter des points d'arrêts, de visualiser des variables, ...

■ Télécommunications – Réseaux

Organisation du réseau local

- Tous les systèmes UNIX supportent le même protocole de télécommunications: `TCP/IP`. Ce protocole est également devenu un standard sur les PC et MacIntosh.

- Toutes les stations de l'observatoire sont reliées entre elles par un réseau **ethernet**, constitué de câbles à paires torsadées, reliés à des *hubs*, à l'intérieur des bâtiments et de **fibre optique** entre ceux-ci.

- Le réseau de l'observatoire est relié au réseau de l'Université Louis Pasteur (**réseau OSIRIS**) par de la fibre optique.

- Le réseau OSIRIS est relié au **réseau INTERNET** au travers de **RENATER** (**RE**seau **NAT**ional pour l'**E**nseignement et la **R**echerche).

Le réseau INTERNET

- Le réseau INTERNET est devenu à partir de 1995 le réseau informatique universel. Après s'être développé dans le monde universitaire, il touche aujourd'hui l'ensemble du secteur commercial, ainsi que les particuliers qui peuvent y accéder en se raccordant à un prestataire de services (Wanadoo, AOL, Compuserve, Worldnet, Free, LibertySurf, etc ...).

- Tout ordinateur sur ce réseau a deux adresses équivalentes:

  ▷ Un **numéro IP**, toujours constitué de 4 champs. Ex: 130.79.128.8

  ▷ Une **adresse INTERNET**, constituée de 2 à 5 champs.
  Ex: `eso.org`, `astrodea.u-strasbg.fr`, `cleese.ipac.caltech.edu`

  ▷ Le lien entre le numéro IP et l'adresse INTERNET est réalisé dans des *serveurs de noms* (**DNS**: **D**omain **N**ame **S**erver).

  ▷ La présence d'une machine sur le réseau peut être vérifiée par la commande:
  ping *numéro-IP* | *nom-INTERNET*

  ▷ La correspondance adresse internet → numéro IP peut être trouvée par la commande:
  nslookup *adresse internet*

- Tout utilisateur d'un ordinateur sur le réseau INTERNET dispose d'une **adresse électronique** unique:
  `username@adresse-internet-de-la-machine`
  Ex: wenger@astro.u-strasbg.fr

Les fontions du réseau INTERNET

- **La connection** à une station du réseau. Depuis une autre station ou depuis un terminal X. Le réseau sert aussi de support au protocole X11, standard de fenêtrage graphique.
- **La mise en réseaux de file systems** au moyen du protocole **NFS**.
- **La messagerie**. Envoie et réception de *courrier*.
- **Les transferts de fichiers**. L'utilitaire **ftp** permet le transfert fiable de fichiers de longueur quelconque entre deux machines du réseau. L'installation sur une machine d'un accès **ftp anonyme** permet l'accès à une arborescence de fichiers sans disposer de compte sur cette machine et de manière sécurisée.
- **L'accès au World Wide Web** (**WWW**) au travers de programmes de navigation (*browsers*) tels que Netscape, Internet Explorer (Microsoft) ou HotJava.

Sécurité

- La taille du réseau INTERNET et sa facilité d'accès comportent certains risques.
- Le principal risque est l'accès frauduleux à une machine et les dégats qui peuvent être occasionnés à cette occasion (destruction de fichiers, passage accès aux privilèges systèmes sur des machines mal protégées, . . .).
- **La principale protection contre les intrusions est le mot de passe de chaque utilisateur**.
- Les intrusions sont aussi rendues possibles par certaines faiblesses du système UNIX:
  - ▷ mauvais paramétrage du système
  - ▷ "trous" de sécurité dues à des erreurs dans certaines commandes sensibles.
    La nature multi-constructeurs du système, sa diffusion dans le monde universitaire et l'existence d'organisme d'alerte (CERT, CIAC) font que ces *bugs* sont rapidement corrigés.
- La nature du système UNIX élimine pratiquement les risques de virus car un utilisateur donné n'a pas accès aux ressources du système, contrairement à un PC ou un MacIntosh.

Connection – ssh, rlogin et telnet

- Par **ssh** pour une connexion sécurisée, dans laquelle les changes sont cryptés. `ssh` permet aussi le *tunelling* qui consiste faire passer par `ssh` des échanges qui passeraient normalement par d'autres protocoles non sécurisés.
- `rlogin` et `telnet` sont deux protocoles qui ne doivent plus êtres utilisés pour des raisons de sécurité. Ils sont d'ailleurs désactivés sur la plupart des machines.
- Quelques commandes utiles à connaître lorsqu'on arrive sur une machine:

`who`	Liste des utilisateurs en session
`w`	Affiche l'activité en cours des utilisateurs connectés
`whoami`	Affiche son nom d'utilisateur courant
`hostname`	Affiche le nom de la machine

Messagerie – mail

- Commande **mail**:
  - ▷ **Lire son mail**: `mail [ -f [`*fic*`] ]`
    - ⋆ Par défaut, les messages sont lus dans le fichier système dans lequel ils sont déposés à leur arrivée (*primary mailbox*).
    - ⋆ Selon la configuration, les messages restent dans ce fichier après lecture, ou sont copiés dans un fichier **mbox**, placé dans le répertoire principal de l'utilisateur (*secondary mailbox*).
    - ⋆ L'option `-f` seule permet de relire le fichier **mbox**.
    - ⋆ L'option `-f` *fic* va lire les messages dans le fichier nommé explicitement.
  - ▷ **Ecrire un mail**: `mail adresse-destinataire ...`
    Taper le mail. Terminer par ^D ou un point seul en première colonne d'une dernière ligne.
  - ▷ **Repondre à un mail** (reply):

    `mail`

    > r *numero-du-message*
    pour répondre à l'expéditeur et aux autres destinataires.

    > R *numero-du-message*
    pour répondre seulement à l'expéditeur.
  - ▷ **Faire suivre un mail à une autre personne** (forward):

    `mail` *nouveau-destinaire*

    Dans le mode composition du message, utiliser la commande "`~f` *numero-du-message*" pour inclure le message à renvoyer.

    Terminer le message et l'envoyer normalement.
  - ▷ **Envoyer un fichier par mail**: `mail [-s "`*subject*`"]` *adr-destinataire* < *fichier*
  - ▷ **Définir des alias**:
    Ecrire dans le fichier "`~/.mailrc`" des commandes du type:
    `alias` *nom liste-d'adresses*

- Principales commandes interactives:

?, **help**	Affiche un résumé des commandes
h n	Liste la page d'en-têtes contenant le message n.
z,**z+**	Liste la page d'en-têtes suivante
z-	Liste la page d'en-têtes précédente
n	Liste le message numéro n
.	Liste le message courant
n	Liste le message suivant
-	Liste le message précédent
vi n	Affiche le message numéro n dans vi
d $n1$-$n2$	Supprime les messages numéros $n1$ à $n2$
u $n1$-$n2$	Rétablit les messages numéros $n1$ à $n2$, supprimés auparavant (par la commande **d**), dans la même exécution de mail
c n *fich*	Copie le message numéro n dans le fichier *fich*
s n *fich*	Copie le message numéro n dans le fichier *fich* et le supprime dans le mail
q	Sort du programme mail en mettant à jour le fichier des messages
x	Sort du programme mail sans toucher au fichiers des mails

m *adresses*	Ecriture d'un mail, envoyé ensuite aux adresses spécifiées.
r *num*	Répond au mail spécifié. La réponse sera envoyée à tous les autres destinataires, en plus de l'éxpéditeur.
R *num*	Répond au mail spécifié. La réponse ne sera envoyée qu'à l'éxpéditeur du message.

- Commandes disponibles en mode saisie de message:

~f *n*	Insère dans le message saisi, le message numéro n. Utile pour renvoyer un message à une autre personne
~s *texte*	Place *texte* dans le sujet du message
~c *nom...*	Rajoute les noms à la liste des destinataires d'une copie du message
~v	Appelle `vi` pour permettre une saisie plus commode du message
^C ^C	Permet d'interrompre un message en cours de saisie (2 fois `^C`)

- La commande `mail` peut être personnalisée par des variables définies dans le fichier `~.mailrc` .

Transfert de fichiers – ftp

- Le transfert de fichiers entre machines UNIX se fait par la commande **ftp** *adresse-de-la-machine*.

- Une fois la connection établie, `ftp` demande le *username* et le *password* de l'utilisateur avec qui on veut transférer des fichiers.

- Dans le cas d'une connection **ftp anonyme**, il faut taper:
  `username:` *anonymous* ou *ftp*
  `password:` *votre-adresse-electronique-complete* afin de vous identifier.

- Les principales commandes de `ftp` sont:

pwd	Affiche le répertoire courant sur la machine distante
cd	Change le répertoire sur la machine distante
ls	Liste le répertoire courant
dir	Liste le répertoire courant d'une façon semblable à `ls -l`
bin	Passe en mode binaire, indispensable pour transférer tout fichier qui n'est pas du texte, notamment les fichiers compressés (ext `.Z`)
get *fich*	Transfert de la machine distante vers la machine locale
put *fich*	Transfert de la machine locale vers la machine distante
mget *fich...*	*Get* multiples. Les fichiers peuvent être définis avec les *wildcards* habituels.
mput *fich...*	*Put* multiples.
prompt	bascule permettant de supprimer et de rétablir la question posée lors des `mget` et `mput`
quit	Quitter `ftp`

- Entre machines UNIX, les fichiers sont souvent transmis en format **tar** et **compressés**, ce qui se traduit par des noms de fichiers se terminant par `.tar.Z` u `.tar.gz` .

  ▷ **tar** est un utilitaire d'archivage de fichiers qui permet de regrouper une arborescence complète de fichiers en un seul. Les trois fonctions utiles sont:

  ⋆ **Créer un fichier tar à partir d'un répertoire et de ses sous-répertoires**:
  `tar cvf fichier.tar répertoire`

  ⋆ **Lister le contenu d'un fichier tar**:
  `tar tvf fichier.tar`
  Les fichiers contenus dans `fichier.tar` sont listés avec leur chemin complet.
  Celui-ci **ne doit pas commencer par un** `/` pour éviter tout problème: en effet, la présence d'un `/` au début signifie que le chemin spécifié est **absolu** (depuis la racine) et a de fortes chances de ne pas être accessible à l'utilisateur ou de risquer d'écraser des fichiers existants.

  ⋆ **Décomposer un fichier tar** en répertoires et fichiers individuels:
  `tar xvf fichier.tar`
  Le nom du fichier peut être remplacé par `-` pour utiliser le fichier standard d'entrée, lors de l'utilisation du `tar` dans un *pipe*.

  ▷ La **compression** et la **décompression** de fichiers sont réalisés par les commandes:

  ⋆ **compress** *fichier ...*: Les fichiers comprimés gardent le même nom avec l'extension `.Z`

  ⋆ **uncompress** *fichier ...*: Les fichiers à décompresser gardent le même nom et perdent leur extension `.Z`

  ⋆ **zcat** *fichier ...*: La sortie décompressée est envoyée en standard output et peut être récupérée au travers d'un *pipe*.

  ⋆ L'extension `.gz` correspond à l'utilitaire `gzip`, plus performant que `compress`:

  · **gzip** *fichier ...*: pour compresser un fichier

  · **gzip -d** *fichier ...*: pour décompresser.

  Une application `pgm.tar.Z` peut être installée d'une des deux façons suivantes:

  ▷
  ```
  uncompress pgm.tar.Z
  tar xvf pgm.tar
  ```

  ▷
  ```
  zcat pgm.tar.Z | tar xvf -
  ```

Mécanisme de base: les sockets

- Toute communication inter-process ou inter-machine dans le monde UNIX utilise un mécanisme unique: **les sockets**.
- Un **socket** est un type particulier de descripteur d'entrées/sorties permettant la communication entre deux process sur deux machines différentes.
- Toutes les applications de type **client/serveur** sont fondées sur ce concept: Le process *client* ouvre une connection avec le process *serveur* et échange ensuite des messages avec lui selon un protocole donné.
- Les fonctions de base sont:
  - ▷ L'ouverture d'un socket (client et serveur)
  - ▷ La mise à l'écoute de clients (serveur)
  - ▷ La connection à un serveur (client)
  - ▷ La lecture de messages transmis (client et serveur)
  - ▷ L'écriture de messages vers le correspondant (client et serveur)
  - ▷ La cloture d'un socket (client et serveur)

X window

Introduction

- X window est un système de gestion de fenêtres graphiques, fonctionnant en client/serveur sur le réseau, permettant à une application sur une machine d'envoyer ses fenêtres de texte, de graphiques et d'images sur n'importe quelle autre machine (terminal) ailleurs sur le réseau. Et aussi de recevoir des commandes du clavier et de la souris depuis ce terminal distant.

- X window a été développé au MIT à partir de 1984. La version la plus récente est X11R6.1

- C'est devenu le standard industriel des terminaux graphiques dans le monde UNIX (terminaux X).

- Le système comprend deux types de programmes:

  - ▷ Des **serveurs X**, programmes gestionnaires de fenêtres, qui gèrent les affichages et les interactions sur une machine possédant un affichage graphique.

  - ▷ Des **clients X**, applications quelconques qui s'exécutent sur n'importent quelle station du réseau et communiquent avec un serveur X pour gérer leur affichage et les événements claviers et souris.

- X window est aussi une librairie permettant de créer des applications X (clients X) exploitant ces possibilités de multi-fenêtrage.

- X window gère:

  - ▷ Des hiérarchies de fenêtres et leur recouvrement

  - ▷ Des opérations à base de caractères et de graphiques

  - ▷ Des affichages sur écrans monochromes ou couleurs

Communication client/serveur

- Un client peut être tout type d'application. Il envoie au serveur **des requêtes**: demandes de création de fenêtre, d'affichage...

- Le serveur gère entièrement un écran. Il communique avec ses clients par l'envoi **d'événements**: frappe d'une touche du clavier ou d'un bouton de la souris, déplacement de celle-ci.

- Le serveur place les requêtes reçues d'un ou plusieurs clients dans une file d'attente et les traite dans l'ordre d'arrivée.

Window Manager

- Un **window manager** est un client particulier dont le rôle est de gérer les fenêtres créees par les autres clients (applications) et affichées par le serveur.
- Il défini l'aspect des fenêtres, la présentation et les fonctionnalités des différentes zones du cadre des fenêtres, la signification des boutons de la souris et gère les menus accessibles sur le fond de l'écran.
- Il permet à l'utilisateur de modifier la position et la taille des fenêtres sur l'écran, de les transformer en icônes et contrôle le recouvrement des fenêtres.
- Il existe de nombreux *window managers*. Chacun a sa propre personnalité (son propre "*look and feel*"): mwm (Motif/OSF), Open Windows (SUN), twm (domaine public), Open Look (AT&T), dxwm (DEC),...

Adressage d'une station d'affichage

- Un serveur X gère une **station d'affichage**.
- Une station d'affichage est définie par un (ou plusieurs) écrans, un clavier et une souris.
- Chaque station d'affichage possède un nom de la forme:
  `machine:affichage.ecran`
  - ▷ `machine` est l'adresse INTERNET de la machine sur laquelle le périphérique est physiquement connecté.
  - ▷ `affichage` est le numéro de l'affichage sur la machine. En général **0**.
  - ▷ `ecran` est le numéro de l'écran à considérer pour le poste d'affichage. Le premier (ou seul) écran est **0**. Il peut alors être omis.

  Une adresse habituelle de poste d'affichage (écran moniteur d'une station ou Terminal X) est de la forme: `machine:0.0`
- Une application client se "*connecte*" à un serveur en définissant l'adresse de l'affichage. Deux méthodes:
  - ▷ En définissant la variable d'environnement `DISPLAY`:
    `setenv DISPLAY cdsxt?:0.0` pour un terminal X à l'observatoire
    En général, cette variable d'environnement est valorisée correctement lors de la connection.
  - ▷ En utilisant l'option `-display` qui est disponible pour toute application X:
    `xappli ... -display cdsxt?:0.0 ...`
- La commande **xhost** permet de gérer la liste des machines à partir desquelles un client pourra se connecter sur le serveur X:

`xhost` $+adresse$	Rajoute une machine à la liste des machines autorisées
`xhost +`	Supprime tout contrôle (autorise tout le monde)
`xhost` $-adresse$	Supprime l'adresse spécifié
`xhost`	Liste toutes les adresses autorisées

Dimensions et positionnement d'une fenêtre

- La taille et le positionnement de la fenêtre d'une application sont contrôlés par l'utilisateur.
- L'origine des coordonnées est X=0,Y=0 et désigne le coin supérieur gauche de l'écran et des fenêtres.
- Des coordonnées définies négativement désignent le coin inférieur droit de l'écran et des fenêtres.
- Le format définissant la *géométrie* d'une fenêtre est:
  `LARGEURxHAUTEUR±X±Y`
  `LARGEUR` et `HAUTEUR` désignent les dimensions de la fenêtre en pixels à l'écran ou en caractères pour certaines applications (`xterm`).
  `+X,+Y` déplacement de gauche à droite, à partir du coin supérieur gauche, en pixels.
  `-X,-Y` déplacement de droite à gauche, avec l'origine dans le coin inférieur droit, en pixels.
- Exemples:

80x24+0+0	fenêtre `xterm` (80x24 colonnes) dans le coin supérieur gauche de l'écran
80x80-0+0	fenêtre `xclock` dans le coin supérieur droit de l'écran

Les polices de caractères

- X window étant un système graphique, il peut afficher des textes en utilisant différentes polices de caractères. Ces polices sont définies dans des fichiers `/usr/lib/X11/fonts/*`.
- Une police de caractères a un nom constitué de 14 champs, représentant toutes ses caractéristiques (famille, type, inclinaison, largeur, résolution, espacement, etc. . .).
  Exemple de nom complet:
  `-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1`
- Une police peut aussi avoir un **alias** simplifiant l'écriture du nom. Les alias sont définis dans des fichiers `fonts.alias` dans les répertoires contenant les polices de caractères.
- Plusieurs commandes particulières permettent de gérer les polices de caractères utilisées dans une application:

xlsfonts	Liste toutes les polices disponibles
xfd -fn *police*	Affiche tous les caractères de la police spécifiée
xfontsel	Permet de selectionner une police de caractères de façon interactive

Les couleurs

- X window permet de spécifier des couleurs sous la forme de trois nombres définissant chacun une couleur primaire: rouge, vert et bleu. L'intensité de chaque couleur va de 0 (couleur absente) à 255 (00–FF en hexadécimal). Une couleur est exprimée sous la forme "*#rrggbb*", chaque couleur étant exprimée par sa valeur hexadécimale.
- X window utilise un fichier contenant des noms de couleurs prédéfinies et qui peuvent être utilisées en lieu et place de la définition numérique (fichier `/usr/lib/X11/rgb`).
- Exemples de couleurs:

noir	000 000 000	#000000	Black
blanc	255 255 255	#FFFFFF	White
rouge	255 000 000	#FF0000	Red
vert	000 255 000	#00FF00	Green
jaune	255 255 000	#FFFF00	Yellow
	123 104 238	#7B68EE	Medium Slate Blue

- Des utilitaires permettent la sélection des couleurs:

`xcolors [-start` *colorname*`]`	affiche la palette des couleurs à partir de la couleur spécifiée
`xcoloredit`	permet la définition de couleurs de façon interactive
`showrgb`	affiche la base de données des couleurs pré-définies

Les widgets

- Toute application X utilise des **widgets** pour réaliser son interaction avec l'écran. Un *widget* peut être une fenêtre, un bouton, une liste, un ascenseur, une boite de dialogue, etc. . .

Les ressources

- Ce sont des attributs (police de caractères, couleurs. . .) définis dans des applications et dont les valeurs par défaut sont stockées dans des fichiers constitués en *bases de données* de ressources. Certaines ressources peuvent aussi être définies sur la ligne d'appel de l'application X.

- Le format d'une ressource est:
  `application[.widget]... .ressource : valeur`
  Une ressource concerne en principe une application
  Les *widgets* successifs désignent des *widgets* imbriqués
  Au plus bas niveau, le *widget* utilise la ressource désignée par sa valeur.

- Il est possible de désigner des ressources génériques en remplaçant plusieurs champs par `*`.
  Exemple: `*foreground : green`
  affectera la valeur `green` à toutes les ressources `foreground` de tous les widgets de toutes les applications.

- La commande **xrdb** permet de gérer la base de données des ressources:

`xrdb -query`	affiche la liste des ressources définies
`xrdb -load` *fichier-de-ressources*	charge un fichier de ressources
`xrdb -merge` *fichier-complementaire*	fusionne le fichier spécifié avec la base existante

- Les ressources d'un utilisateur seront traitées automatiquement si elles sont dans le fichier `~/.xresources` ou `~/.Xresources` en général.

- Exemple:

```
Xbiff*Geometry: 60x60+938+79
Xterm*Font: 8x13
Xterm*BoldFont: 8x13bold
Xload*BorderWidth: 0
```

Démarrage d'une session X

- L'application **xdm** (*display manager*) gère hors session l'affichage d'une fenêtre d'accueil sur le terminal X
- Lorsqu'un utilisateur tape son nom et son mot de passe, `xdm` exécute le script spécifié dans sa ressource "`DisplayManager*session`".
- En général, ce script procède à diverses initialisations et lance un script "`.xsession`" dans le répertoire principal de l'utilisateur.
- Ce script termine les initialisations par les spécificités de l'utilisateur. Entre autre il va rajouter les ressources de l'utilisateur (fichier "`.xresources`"). Puis il lance le *window manager*.
- Le *window manager* a ses propres fichiers d'initialisation: ressources, définitions des menus de fond d'écran, fenêtres `xterm` de démarrage ou toute autre application.
- L'utilisateur peut ensuite commencer à travailler...

Index