UTILISER UNIX

Philippe Marquet

marquet@lifl.fr
http://www.lifl.fr/~marquet/ens/uu
Laboratoire d'informatique fondamentale de Lille
Université des sciences et technologies de Lille
France

Licence d'informatique

créé janvier 1994
dernière révision septembre 2001

PRÉLIMINAIRE

Table des matières

Remerciements

Merci à

- Fabien COELHO
- Philippe DURIF
- Jean-François MÉHAUT
- Jean-François ROOS

Bibliographie

- Christian Pélissier
  Utilisation et Administration du Système Unix
  Hermès, 1992 ; 3e éd., 1998
- Jean-Marie Rifflet
  La Programmation sous Unix
  McGraw-Hill, 1986 ; Ediscience International, 3e ed., 1993
- Don Libes, Sandy Ressler
  Life with Unix
  Prentice-Hall, 1989
- Gail & Paul Anderson
  The Unix C Shell Field Guide
  Prentice-Hall, 1986

INTRODUCTION — HISTORIQUE & PHILOSOPHIE

Contents

Unix, un système d'exploitation

- Système d'exploitation
  - Gestionnaire de ressources, ou
  - Machine abstraite

  (– Cf. cours de maîtrise)
- Système d'exploitation temps partagé

  Système d'exploitation multi-utilisateur
  - Plusieurs utilisateurs simultanés
  - Sur une même machine
  - Impression (fausse) qu'ils ont une machine chacun
- Système d'exploitation multi-tâche
  - Plusieurs exécutions de programmes
  - En parallèle ou pseudo-parallèle
- Unix est interactif
- Premières versions développées en 1969
  - Ken THOMPSON : Unix
  - Dennis RITCHIE : C

- Écrit dans un langage évolué
  - 95 % en C
  - 5 % assembleur
    Partie dépendante du matériel
    Ex : manipulation des registres du processeur
  - Portage Unix = compilateur C + courage (pour les 5 %)
  - Initialement, sources domaine public
- Langage évolué ⇒
  - Maintenance plus efficace
  - Interface d'accès au système naturelle
  - Appel système ≡ appel de fonctions
- Unix est un système portable
  - Processeurs : Intel 80x86, Motorola 680x0, Sparc, MIPS, Alpha...
  - Constructeurs : IBM AIX, DEC Ultrix, DEC OSF, HP HPux, Sun SunOs, Sun Solaris...

Système portable ⇒

- Système ouvert ≠ système propriétaire
  - Une gamme de machine/un constructeur
  - IBM : VM, MVS, OS400
    BULL : GCOS
    CDC : NOSVE
    DEC : VMS
    HP : MPE
- Large variété de matériels
  - Du PC aux super-calculateurs
  - Initialement : mini-ordinateur PDP DEC
    Idée : développer des concepts disponibles sur gros systèmes (Multics sur Honneywell)
  - Évolution : toute la gamme de machines
    PC : Xenix, Linux
    Cray : Unicos
    Machines parallèles : CM-5
    ...

- Percée : stations de travail
  - Apollo, Sun...
  - Station de travail ?
    Processeur puissant
    Écran graphique : multifenêtrage
    Interface réseau (local) : Ethernet
  - Système d'exploitation : Unix
  - Environnement graphique : X-Windows
- Conçu par des développeurs pour des développeurs
  Outils disponibles de base
  - Éditeurs de texte `ed`, `vi`
  - Compilateur C
  - Outils de génie logiciel
    `make`
    `sccs`
  - Debugger, mesure de performances de programmes
  - Générateurs d'analyseurs lexicaux et syntaxiques
  - Interpréteurs de commandes puissants et programmables

- Arborescence du système de fichiers
  - Repris par de nombreux systèmes d'exploitation actuels
  - Fichiers spéciaux pour les périphériques
  - Exemple : imprimante
  - Entrées/sorties indépendantes des périphériques
- Introduction de nombreux concepts
  - Simplicité de ces concepts
  - Repris par d'autres systèmes
- Structure en couches d'Unix

User (programmes utilisateurs)
System program (compilateurs, shells...)
Kernel (file, process, information)
Hardware (cpu, register, disk, terminal)

Histoire

- Première version
  1969
  Laboratoire Bell de la société AT&T
- Ken Thompson
  - Contribution au projet Multics, 1968
  - Possibilités avancées, gourmand en ressource
- Développement d'un projet propre
  - Ken Thompson, Dennis Ritchie, Joseph Ossanna
  - Simulation du mouvement de corps dans le système solaire
  - "*Little-used PDP-7 sitting in a corner*"
  - Mise en place d'un environnement *à la* Multics
  - Écrit en assembleur
  - Concepts de base d'Unix :
    - ▷ Multiprocessing (`fork()`)
    - ▷ Système de fichiers (i-nodes, répertoires, fichiers spéciaux)
    - ▷ Multics → Unics → Unix

- Autre projet
  - Système de "text-processing"
  - Portage d'Unix sur PDP-11
  - Résultat : `roff` (`runoff`, `troff`) + éditeur de texte !
- Un bureau des Bell Labs était à la recherche d'un système de traitement de textes
  - Sélection des travaux du groupe
  - Premiers utilisateurs officiels d'Unix
  - Documentation du système "*First Edition*", nov. 1970
  - Numérotation suivant cette documentation
- Langages
  - B est un langage conçu par Thomson et Ritchie, origine dans BCPL (← CPL ← Algol60)
  - Réécriture du noyau en NB, version modifiée de B
  - NB devient C
  - Expérience de Multics : PL/1
  - 1972, version 2, réécriture du noyau et du shell en C

- 5[e] édition, 1974
  - Présentation dans "*Communications of the ACM*"
  - Disponibilité officielle
  - Distribution des sources "*for educational purposes*"
  - Contact avec les "utilisateurs" — `uucp` mail, phone to Ken and Dennis
  - The `research` group
- PWB
  - Autre groupe au sein des Laboratoires Bells
  - Utilisation de Unix pour de gros développements
    ⇒ conception de `sccs`
  - Propre distribution d'Unix
    Au sein et à l'extérieur des Bell Labs
- 6[e] édition, 1975
  - Première version commercialisée
  - Toujours pas de support
  - Portage sur PDP-11/70 par un groupe de recherche
    Université de Californie, Berkeley
    Redistribution par Berkeley d'Unix et de ses modifications

- Version 7, 1979
  - K&R complet
  - Steve BOURNE shell `sh`
  - Portage sur un VAX → Unix 32V
- 3 branches d'Unix
  Issues de Unix 32V
  1. BSD : Berkeley Software Distribution
  2. V : Bell Labs research group
  3. System : Commerciale AT&T
- System V, 1983
  - Version commerciale AT&T
  - Intégration de développement BSD
  - Pour la première fois : support AT&T

  ⇒ Confiance du marché

Arbre généalogique

	OSF	Berkeley	Bell Labs	AT&T
1970			V1	
...				
1975			V6	
1976				
1977		BSD 1.0		
1978		BSD 2.0	V7	
1979		BSD 3.0	Unix 32V	
1980		BSD 4.0		
1981		BSD 4.1		
1982				System III
1983		BSD 4.2		System IV
1984				System V R1
1985			V8	System V R2
1986		BSD 4.3		System V R3
1987				
1988	MACH			
1989		BSD 4.2 Tahaoe		System V R4
1990				
1991	OSF 1	BSD 4.3 Reno		
1992				
1993				
...		BSD 4.4		System V R5

Versions commerciales d'Unix

- Amalgame des versions BSD4.3 et SVR5
- Apports de BSD à la Version 7
  - Gestion de la mémoire virtuelle
  - Mécanisme de communication inter-processus plus performant (IPC)
  - Support d'ethernet TCP/IP (ftp, telnet, r-commandes)
  - Système de fichiers plus performant (quotas)
  - Introduction du C-shell
  - Commandes plus nombreuses
- Apports de System V à la Version 7
  - Les IPC (Inter Processus Communication) : sémaphores, mémoire partagée, messages, fifo nommées
  - Amélioration de la commande `at` : possibilité de batch...
  - Généralisation de `cron` pour tous les utilisateurs
  - Variable `CDPATH` des shells
  - Introduction du Korn shell `ksh`
  - Introduction des bibliothèques partagées
- Chacune des branches introduit de nouvelles options dans des commandes Version 7

- OSF
  - Open Software Fondation
  - Issu de Mach
    Version moderne d'Unix
    Support de processeurs multiples et du parallélisme
  - Distributeur de licences Unix
- GNU de la FSF (Free Software Foundation)
  - GNU's Not Unix
  - But : fournir un système Unix gratuit
  - Déjà bien avancé — Programmes très performants
  - Le célèbre et incontournable GNU `emacs`
  - Compilateurs C et C++, `gcc`, `g++`
  - Outils `as`, `ar`, `dbx`...
  - Utilitaires Unix : `make`, `rcs`...
- Ultrix
  - DEC — Digital Equipment Corp.
  - Initialement basé sur BSD 4.2 (Ultrix 1.1)
  - Reprise de fonctionnalités SVR2 (1.2)
  - Apparition de X-windows sur VaxStation
  - Reprise des logiciels NFS et NIS de Sun (2.0)
  - Niveau de sécurité accru (4.0)
  - Support de machines multi-processeurs (4.0)
  - Conforme aux normes d'internationalisation des caractères de l'ISO

- SunOS, Solaris
  - Sun
  - Dérivé de BSD 4.2
  - Introduction de NFS — Network File System
    Système de fichiers distribués
  - Introduction de NIS, ex "yellow pages"
    Network Information Services
    Gestion centralisée de l'administration
  - Évolution vers System V (édition de lien dynamique)
  - Environnement graphique SunView → OpenLook (X11R4)
- Linux
  - Intel 80x86, PC (à l'origine)
  - Commencé par Linus TORVALD en 1991.
    Inspiré de minix (Andrew TANENBAUM)
  - "Free software" / "Open-Source" (gratuit et libre)
  - Linux est Unix
  - Linux est robuste, stable, bien supporté, bien diffusé...
- Standardisation d'Unix
  - Deux organisations de normalisation :
    1. Association européenne X-Open
    2. Groupe POSIX, membre de l'IEEE
       Portable Open System Interface eXchange
       Association of Electrical and Electronic Engineers
  - Conformité aux normes POSIX

SYSTÈME DE FICHIERS

—

COMMANDES DE BASE

Contents

Unix

- Système d'exploitation
  Interface architecture/applications
- De plus en plus répandu
  PC $\longrightarrow$ super-calculateurs
- Fort lien avec le langage C
- Philosophie Unix

Utilisateur

- Nom d'utilisateur : `marquet`, `duchmol`
- Mot de passe
  - `passwd`
    (`yppasswd` ou `nispasswd`)

```
% passwd
Changing NIS password for marquet on homel.
Old password:
New password:
Retype new password:
NIS entry changed on homel
```

  - Sécurité
  - Responsabilité
- Appartient à un groupe : `enseign`, `licence`
- Chez-soi : `/home/enseign/marquet`
- Connexion ou login

```
SunOs Unix (homel)
login: marquet
Password:
%
```

  (Session graphique)
- Après le login : interpréteur de commandes (`csh`)
- Quitter :

  `logout` ou `exit`

  Control-D `^D`

Interpréteur de commandes

- Shell

1 Interface utilisateur pour lancer des commandes

2 Exécution de scripts ($\equiv$ programmes shells)

- Différents shells :

2 $\Rightarrow$ Bourne shell `sh` (ou `bash`)
  - Korn shell `ksh`

1 $\Rightarrow$ C-shell (`csh`, `tcsh`)
  - `tcsh`
  - `perl`

- Contrôle des processus
- Substitutions diverses
- Langage de contrôle
- Services divers

Syntaxe d'une commande

`nom [option]... [arguments]...`

- Premier mot : nom de la commande
- Puis les options (-...)

  `-la` ≡ `-l -a`
- Et les arguments
- Interactif : prompt string $

```
$ echo hello       world
hello world
$ ls -F
file
$ rm -i file
rm: remove file? y
$ cd ..
$
```


Documentation

- Manuel en ligne : `man`

```
$ man man
  ...
```

`man [section] command`

- Section : précise où chercher
  - 1 : users commands (`man 1 ls`)
  - 2 : system calls (`man 2 chmod`)
  - 3 : subroutines (`man 3 fprintf`)
  - 4 : devices (`man 4 NFS`)
  - 5 : file formats (`man 5 passwd`)
  - 6 : games (`man 6 chess`)
  - 7 : miscellaneous (`man 7 man`)
  - 8 : syst adm (`man 8 boot`)
  - l, n : local, new...
- `man ? List`

  liste des commandes de la section ?

  Courte description
- Interface graphique : `xman`
- Commande `info`

Parties d'un manuel en ligne

- NAME
- SYNOPSIS
- AVAILIBILITY
- DESCRIPTION
- OPTIONS
- EXAMPLES
- WARNINGS
- FILES
- SEE ALSO
- DIAGNOSTICS
- BUGS

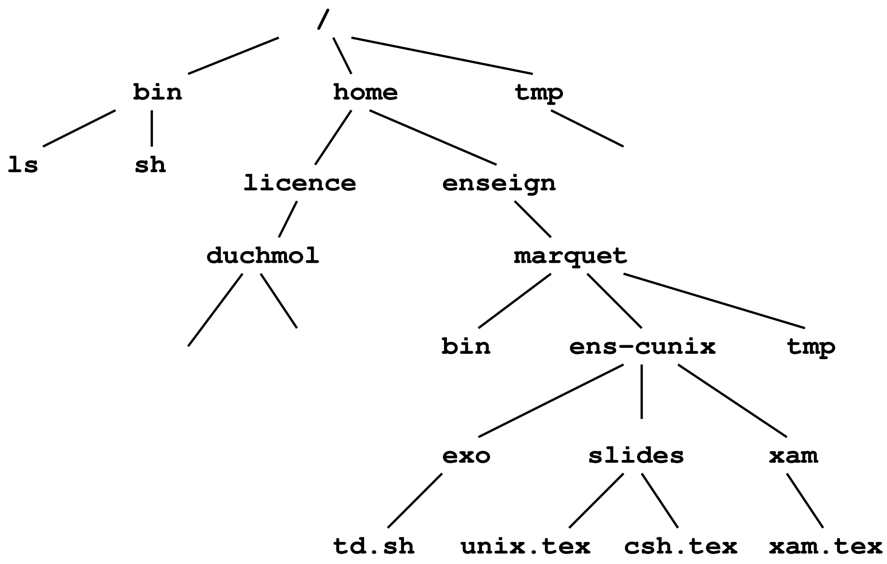
Fichier Unix

- Stockage des données
- Hiérarchisé, arborescence de répertoires
- Format : séquence d'octets/de caractères
- Contenu :
  - Programmes exécutables, shell-scripts, sources, textes
  - Répertoires
  - Liens
  - Périphériques (/dev/*)
  - Fichiers spéciaux

Arboréscence de répertoires

- Un fichier est dans un répertoire
- Répertoire racine /
- Chaque répertoire a un père unique (sauf /)

`/home/enseign/marquet/ens-cunix/slides/unix.tex`



Répertoires généraux

- `/sbin` : commandes minimales
- `/bin` : commandes principales
- `/dev` : périphériques
- `/etc` : configuration et administration système
- `/home` : utilisateurs (ou `/users`)
- `/var` : variable, à garder...
- `/tmp` : temporaire
- `/usr` : commandes locales
- `/lib` : librairies
- `/usr/doc` et `/usr/share/doc` : documentation

Chemin d'accès à un fichier

- Suite des noms des répertoires intermédiaires
- Séparés par /
- Sens normal : racine → feuille

  `ens-cunix/slides/unix.tex`

  `/usr/local/bin/ftp`

- Absolu : commence par /
- Sinon relatif : à partir du répertoire courant
- En `csh` et `tcsh`
  ˜ : répertoire de login propre
  `˜username` : d'un utilisateur
- `.` : répertoire courant (forcer une commande)
- `..` : répertoire père
- Problème des liens symboliques

Syntaxe d'un nom de fichier

- Nom terminal d'un fichier ("basename")
- Suite de caractères ASCII sauf le /
  Terminée par un \0
- Longueur limitée (14, ou 255 caractères...)
- Pas de contraintes, mais des suffixes conventionnels
- Les langages

`.s`	Assembleur
`.c`, `.h`	C (et include)
`.java`, `.class`	Java
`.f`	Fortran
`.el`, `.elc`	Lisp (compilé)
`.l .y`	Lex et Yacc
`.o`	Binaire objet
`.a`	Librairie de binaires objets

- L'absence de suffixe est généralement réservée aux fichiers exécutables et aux répertoires

- Documentation

`.html`, `.htm`	HTML
`.1` à `.8`	Manuel en ligne (`nroff`)
`.texi`	Commande `info`
`.ps`	PostScript
`.pdf`	Portable Document Format
`.tex`	TeX ou LaTeX

- Divers

`.Z`	Compressé avec `compress`
`.gz`	Compressé avec `gzip`
`.bz2`	Compressé avec `bzip2`
`.tar`	Archive `tar`
`.tgz`	Archive GNU `tar` compressée
`.rpm`	RPM Package Manager
`.uu`	Fichier “uucodé” (`uuencode`)
`.old` `.bak` `~`	Version précédente

Méta-caractères

- Substitutions réalisées par le shell
- Expressions régulières pour les noms de fichiers

`?`	un caractère quelconque (y compris le `.`)
`[list]`	un caractère quelconque de la `list`
`[lower-upper]`	un caractère quelconque entre `lower` et `upper`
`*`	tout motif (y compris le vide)

- Protection d'un caractère : `\`

Exemple

```
homel% ls /bin/t*
/bin/t300       /bin/tcov       /bin/time       /bin/traffic
/bin/t300s      /bin/tee        /bin/tip        /bin/troff
/bin/t4013      /bin/tek        /bin/toolplaces /bin/true
/bin/t450       /bin/tektool    /bin/touch      /bin/tsort
/bin/tar        /bin/test       /bin/tr         /bin/tty
/bin/tbl        /bin/textedit   /bin/trace

homel% ls /bin/t???*
/bin/t300       /bin/tcov       /bin/time       /bin/traffic
/bin/t300s      /bin/tektool    /bin/toolplaces /bin/troff
/bin/t4013      /bin/test       /bin/touch      /bin/true
/bin/t450       /bin/textedit   /bin/trace      /bin/tsort

homel% ls /bin/t[0-9]*
/bin/t300   /bin/t300s  /bin/t4013  /bin/t450

homel% ls /bin/t*[rt]
/bin/tar       /bin/test          /bin/textedit
/bin/tr        /bin/tsort
```


Manipulation des répertoires

`pwd`

- Affiche le répertoire courant

`cd [dirname]`

- Change le répertoire courant

  `cd` : va au répertoire de login

  `cd ~` : idem

  `cd ~marquet` : idem

  `cd /home/enseign/marquet` : idem

  `cd ../..` : remonte de deux répertoires

- Par défaut : répertoire de login

`mkdir [-p] dirname...`

- Création d'un répertoire

  `mkdir /home/enseign/marquet/tmp ~/foo`

- L'option `-p` crée les répertoires pères si nécessaire

```
homel% ls
titi
homel% mkdir foo/bar
mkdir: foo/bar: No such file or directory
homel% mkdir -p foo/bar
homel% ls
gee  foo
homel% ls foo
bar
```

`rmdir dirname...`

- Effacer un répertoire

  `rmdir ~/foo`

- Il doit être vide (cf. `rm -r`)

Caractéristiques d'un fichier

- Propriétaire : un utilisateur
- Dans un groupe
- Date (dernière modification, sauvegarde)
- Droits d'accès (ugo : user, group, others)
- Type (fichier, répertoire, lien...)
- I-node

```
$ ls -lg /vmunix
-rwxr-xr-x 1 root bin 5854544 Sep 01 09:11 /vmunix
```

```
$ ls -lg /vmunix
-rwxr-xr-x 1 root bin 5854544 Sep 01 09:11 /vmunix
```

Premier champ : 10 caractères

- 1er caractère
  Type du fichier
  - `-` Fichier normal
  - `d` Répertoire
  - `l` Lien symbolique
- Caractères 2 à 10
  Droits
  user, group, others : $3 \times 3 = 9$ caractères
  `rwx-`
  1. Fichier simple :
     - `r` lecture
     - `w` écriture (modification)
     - `x` exécution
       `s` *(suid)* exécution comme le propriétaire
  2. Répertoire :
     - `r` lecture (`ls`)
     - `w` modification (`mv`, `rm`, `cp`)
     - `x` passage


```
$ ls -lg /vmunix
-rwxr-xr-x 1 root bin 5854544 Sep 01 09:11 /vmunix
```

- Deuxième champ
  - Entier
  - Nombre de liens (physiques) vers le fichier
- Troisième champ
  - Propriétaire du fichier
- Quatrième champ
  - Groupe du fichier
- Cinquième champ
  - Taille en octets (caractères) du fichier
- Champs suivants — date de modification du fichier
  - Mois
  - Quantième
  - Heure (ou année)
- Nom du fichier

Contenu d'un répertoire

`ls [-algiARF...]  [name]...`

- Options

  `-a` : all (même commençant par un `.`)

  `-A` : All (idem que `-a`, sauf `.` et `..`)

  `-l` : format long

  `-g` : groupe

  `-i` : i-node du fichier

  `-d` : si l'argument est un répertoire, écrit seulement son nom et ne liste pas son contenu

  `-f` : force l'argument à être un répertoire, et liste les noms trouvés

  `-c` ou `-t` : tri par dernière date de modification

  `-R` : récursif

  `-F` : précise le type par un suffixe (`xx/ xx* xx@ xx=`)

- Nom de fichier ou de répertoire facultatif

Exemples

```
homel% ls /bin
X11    grep   roffbib  ...

homel% ls -F
bin@            export/         mnt/     ...
cdrom/          home/           net/     ...
dev/            kadb*                    ...

homel% ls -a
.               bin             kadb     ...
..              cdrom           lib      ...
.cshrc          dev             mnt      ...
.login          etc             net      ...
```

Manipulations des fichiers

```
mv [-] [-fi] filename1 filename2
mv [-] [-fi] dirname1 dirname2
mv [-] [-fi] filename...  dirname
```

- Déplace ou renomme un fichier/un répertoire
- `mv filename1 filename2`
  Renomme `filename1` en `filename2`
- `mv dirname1 dirname2`
  Si `dirname2` n'existe pas,
  Renomme `dirname1` en `dirname2`
- `mv filename... dirname`
  Place les fichiers `filename` dans le répertoire `dirname`
- Les options :
  - `-` Les arguments suivants sont des noms de fichiers
  - `-i` Mode interactif
  - `-f` Force

```
homel% mv /home/licence/* /home/maitrise
```



```
cp [-ip] filename1 filename2
cp -rR [-ip] dirname1 dirname2
cp [-iprR] filename...  dirname
```

- Recopie de fichiers
- `cp filename1 filename2`
  Recopie du fichier `filename1` dans le fichier `filename2`
- `cp -rR [-ip] dirname1 dirname2`
  Copie récursive de `dirname1` dans `dirname2`
- `cp [-iprR] filename... dirname`
  Copie des fichiers `filename` dans `dirname`
  `dirname` doit déjà exister
- Options :
  - `-i` Mode interactif
  - `-p` Duplique les droits et la date de modification
  - `-r` ou `-R` Copie récursive

```
rm [-] [-fir] filename...
```

- Destruction de fichiers
- Options :
  - `-` Fin des options
  - `-r` Destruction récursive
  - `-i` Mode interactif
  - `-f` Force

Visualisation/Édition des fichiers

```
cat [-] [-bnv..]  [filename]...
```

- Affiche le contenu des fichiers
- Par défaut : l'entrée standard
- Concatène les fichiers
- Options
  - `-b` ou `-n` Numérotation des lignes
  - `-v` Filtre et affiche les caractères non imprimables
    Par exemple : `^C` (Control-C)...

```
more [options] [filename]...
less
```

- Affiche en mode page/page les fichiers
- Affiche l'entrée standard par défaut

```
homel% more un-tres-gros-fichier
...
```

```
head [-number] [filename]...
tail [+|-number] [filename]...
```

- Affiche les premières ou dernières lignes d'un fichier
- Options
  - `-number` Affiche les `number` premières/dernières lignes
  - `+number` Commence l'affichage à la ligne `number`

```
ed filename
vi [filename]...
textedit filename
kedit [filename]...
emacs [filename]...
```

- Éditeurs de fichiers
- `ed` Éditeur ligne
- `vi` Éditeur de texte pleine page
- `textedit` Éditeur SunView ; `kedit` Éditeur KDE
- `emacs` Éditeur programmable, extensible
  Produit GNU, un must

Liens symboliques et physiques

```
ln [-s] filename [linkname]
```

- Fichier de lien sur un autre fichier (ou répertoire)
- Plusieurs noms pour un même objet
- Deux types de liens
  1. Par nom, lien symbolique
     `ln -s filename [linkname]`
  2. Par i-node, lien réel
     `ln filename [filename-prime]`
- Exemple

```
$ pwd
/home/enseign/marquet
$ ln -s /usr/bin mon-lien-vers-usr-bin
$ ls
mon-lien-vers-usr-bin ...
$ cd mon-lien-vers-usr-bin
$ pwd
/usr/bin
$ cd ; ln -s /etc/hosts mon-fichier-host
$ more mon-fichier-host
...
```

Différence lien symbolique/lien physique

- Lien physique

```
$ more file1
abcdef
$ ln file1 file2
$ more file2
abcdef
$ rm file1
$ more file2
abcdef
```

- Lien symbolique

```
$ more file1
abcdef
$ ln -s file1 file2
$ more file2
abcdef
$ rm file1
$ more file2
file2: No such file or directory
```


Changement des caractéristiques d'un fichier

Les permissions

```
chmod [-R] [ugoa] [+-=] [rwx st ugo] filename...
chmod [-R] octalmode filename...
```

- Changement des droits
- Option `-R` : récursif
- Deux spécifications possibles du mode : symbolique ou octal

- Symbolique
  - a appliqué à tous (défaut)
  - u appliqué au propriétaire (user)
  - g appliqué au groupe
  - o appliqué aux autres (others)

  - + ajout de droits
  - - retrait de droits
  - = positionnement de droits

  - r droit de lecture (read)
  - w droit d'ecriture (write)
  - x droit d'exécution (ou de traverser un répertoire)
  - s « set-uid » bit (associé à u)
    « set-gid » bit (associé à g)
  - t « sticky » bit

  - ugo recopie des droits du propriétaire, groupe, ou autre

- Ou en octal

  Le mode est spécifié par combinaison des nombres octaux

4000	positionne le « set-uid » bit
2000	positionne le « set-gid » bit
1000	positionne le « sticky » bit
400	lisible par le propriétaire
200	modifiable par le propriétaire
100	exécutable par le propriétaire
40	lisible par les membres du groupe propriétaire
20	modifiable le groupe propriétaire
10	exécutable le groupe
4	lisible par les autres utilisateurs
2	modifiable par les autres utilisateurs
1	exécutable par les autres utilisateurs

- Le « sticky » bit
  - Pour un fichier : limite les swap entre différentes exécutions
  - Pour un répertoire, mode `drwxrwxrwt`
    Seul le propriétaire d'un fichier de ce répertoire (et root) peuvent détruire le fichier
    `/tmp`, `/usr/spool/mail`...

Exemple

```
$ ls -lgF file
-rw--w---- 1 marquet enseign 0 Jan  4 10:17 file

$ chmod a+x,g-w+rs file

$ ls -lgF file
-rwxr-s--x 1 marquet enseign 0 Jan  4 10:17 file*
```

- Fichier exécutable par tous
- Avec les droits du groupe `enseign`
- Lisible par le groupe
- Modifiable par le propriétaire

```
$ chmod 2751 file
```


`umask [nnn]`

- Définit (ou affiche) les droits des fichiers à leur création
- Argument : masque exprimé en octal (défaut : 022)
- Fichiers : $\forall$ masque, jamais de `x`, sauf éditeur de liens
- Définition du masque

r w x	r - x	- - -	protection symbolique
1 1 1	1 0 1	0 0 0	protection binaire (750 en octal)
0 0 0	0 1 0	1 1 1	masque binaire
0	2	7	masque octal

```
homel% umask 027
homel% umask
027
homel% cp /dev/null foo ; mkdir bar
homel% ls -ld foo bar
drwxr-x---  2 marquet       512 Jan 27 11:35 bar
-rw-r-----  1 marquet         0 Jan 27 11:35 foo
```

Les propriétaires

```
chown username filename...
chgrp groupname filename...
```

- Change le propriétaire ou le groupe propriétaire des fichiers
- Être le propriétaire ou root (System V)
- Être root (BSD 4.2, quotas disque)
- Retirent les bits s

Espace disque

```
df [filesystem]... [filename]...
```

- Information sur l'espace disque du système

```
homel% df
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/sd0a              22603   13534    6809    67%    /
/dev/sd0g             139927  123404    2531    98%    /usr
/dev/sd3h             299621  243555   26104    90%    /soft
/dev/sd2a            1255494 1091348   38597    97%    /home

ms2% df
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/sd0a               7735    5462    1500    78%    /
/dev/sd0g             151399  131218    5042    96%    /usr
homel:/home          1255494 1091367   38578    97%    /home
homel:/soft           299621  243555   26104    90%    /soft
homel:/var/spool/mail  22603   13536    6807    67%    /var/spool/mail
```

`du [-s] [-a] [filename]...`

- Taille disque utilisée
  par un fichier
  par un répertoire (récursif)
- En kilo-octets
- Options :
  - `-s` Seulement le total pour chaque argument
  - `-a` Toutes les entrées, sous-répertoires...

```
% pwd
/home/licence/duchmol
% du
5    ./jokes
33   ./squash
44   ./tech.papers/lpr.document
217  ./tech.papers/new.manager
401  ./tech.papers
144  ./memos
80   ./letters
388  ./window
93   ./messages
15   ./useful.news
1211 .
% du -s .
1211 .
```


`quota -v`

- Reporte les quotas et les limites
  Place disque
  Taille
  Nombre de fichiers
- Propriétaire des fichiers

```
home1% quota -v
Disk quotas for marquet (uid 7314):
Filesystem     usage  quota  limit    timeleft //
/export/home   45214  50000  50500             //

         //  files  quota  limit    timeleft
         //  2010      0      0

home1% quota -v
Disk quotas for marquet (uid 7314):
Filesystem     usage  quota  limit    timeleft //
/export/home   50491  50000  50500    7.0 days //

         //  files  quota  limit    timeleft
         //  2011      0      0
```

Type d'un fichier

`file files...`

```
$ pwd ; file .* *
/
.cshrc:          ascii text
.login:          c-shell commands
.profile:        shell commands
bin:             symbolic link to usr/bin
home:            directory
kadb:            sparc executable not stripped
```


INTERPRÉTEUR DE COMMANDES

Contents

Interpréteur de commandes

- Shell
- Interface utilisateur pour lancer des commandes

(• Exécution de scripts)

- Combinaisons de commandes
- Contrôle des processus
- Substitutions diverses

  Variables

  Alias

  ...

Pourquoi le C shell

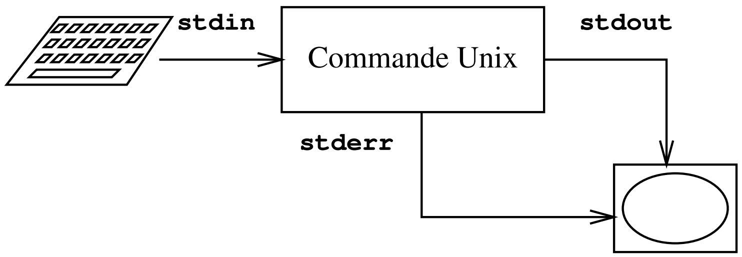
- Nombreux shells : Bourne shell, C shell, Korn shell...
- C shell : aspects interactifs
  - Alias
  - Historique des commandes
  - ...
- Bourne shell : programmation

Ligne de commandes

- Minimale :

  `% commande [options] [arguments]`

- "*Return*" ⇒ modifications possibles
  - Control-H
    Control-U
    Control-R
  - `tcsh` : édition possible
- Exécution de la commande
  Entrée standard, Sortie standard } associées au terminal



Commandes multiples

- Différentes commandes sur une ligne

```
% commande1 ; commande2 ; ... commanden
```

- Exécution séquentielle
- Processus indépendants
- Exemples

```
% mkdir proposal ; cd proposal
```

```
% cd utils ; cc -o gets gets.c ; cd ..
```


Groupe de commandes

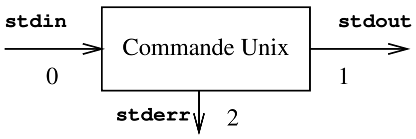
- Exécuter de multiples commandes
- Sans affecter l'environnement courant
  Répertoire courant, variables...
- Création d'un sous-shell
- Exemple :

```
% (cd utils ; cc -o gets gets.c )
```

- Un groupe de commandes peut remplacer n'importe quelle commande

Redirection des entrées/sorties

- Une commande ouvre 3 descripteurs de fichiers



- Une telle commande est appelée *filtre*
- Rediriger ces entrées et sorties vers une autre commande ou un autre fichier
- C shell (`csh`)

<	redirige l'entrée standard
>	redirige la sortie standard
>&	redirige la sortie standard et la sortie d'erreur
>>	concatène la sortie standard
>>&	concatène la sortie standard et la sortie d'erreur

Exemple

```
homel% pwd
/home/enseign/marquet
homel% pwd > pwd.out
homel% cat pwd.out
/home/enseign/marquet
homel% pwd >> pwd.out
homel% cat pwd.out
/home/enseign/marquet
/home/enseign/marquet
```

cat un éditeur !

```
homel% cat > foo
Ici je tape le contenu de foo
Je finis par un Control-D en debut de ligne
^D
homel% cat foo
Ici je tape le contenu de foo
Je finis par un Control-D en debut de ligne
homel% cat >> foo
Voici la suite de foo
^D
homel% cat foo
Ici je tape le contenu de foo
Je finis par un Control-D en debut de ligne
Voici la suite de foo
homel%
```


Redirection des erreurs

```
% ls -l bar > foo
bar not found
% cat foo
% ls -l bar >& foo
% cat foo
bar not found
```

Redirection de commandes multiples

```
% date ; cc -o gets gets.c >& compil.out
Mon Jan 17 15:22:06 MET 1994
% ( date ; cc -o gets gets.c) >& compil.out
```

Séparation de stdout et de stderr

```
% cat calendar logfile >& save
% cat save
Oct 14 Planning meeting
Oct 21 Budget meeting
cat: can't open logfile
% (cat calendar logfile > save) >& errfile
% cat save
Oct 14 Planning meeting
Oct 21 Budget meeting
% cat errfile
cat: can't open logfile
```


(Digression : la commande who)

- Affiche la liste des utilisateurs connectés

```
% who
marquet        tty1    Sep 24  6:39
duchmol        tty3    Sep 29 15:16
marquet        tty2    Sep 24  7:14
```

Protéger les redirections

- Une redirection peut détruire le contenu d'un fichier

```
% who > who.log
```

- Une concaténation peut créer un fichier

```
% date >> statis
```

L'information est écrite dans `statis` et non `status`

- Variable `noclobber` du C shell

```
% set noclobber
% who > who.log
who.log: File exists
% date >> statis
statis: No such file or directory
```

- Dépasser `noclobber`
(Pas d'espace avant le !)

```
% set noclobber
% ls
status   who.log
% who >! who.log
% date >>! statis
```

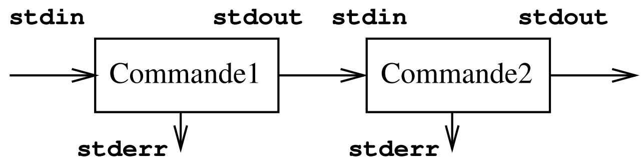

Connecter des commandes : les pipes

- Combien d'utilisateurs connectés ?

```
% who > temp ; wc -l < temp ; rm temp
22
```

- Connecter deux commandes par un "pipe"

```
% who | wc -l
22
```



- Forme générale :

```
commande1 | commande2 | ... | commanden
```

- Autre exemple :

```
% cat dict1 dict2 | epelle -french | sort \
  > dict.spell-err
```

- Passage des erreurs via les "pipes"
Utilisation de `|&`
Peu utilisé

Exécution conditionnelle

- Exécution d'une commande si et seulement si une commande précédente
  - s'est terminée par un succès
  - ou par un échec
- Exemple : impression si succès

```
% grep Aug.92 transac > costs && lpr costs
```

- Deux opérateurs

Succès de la 1^re^commande	Opérateur	Exécution de la 2^e^commande
oui	||	non
non	||	oui
oui	&&	oui
non	&&	non

- Autre exemple

```
% cc tester.c &> err || mail marquet < err
```


Les variables

- Variables d'un langage de programmation
- Contient une chaîne de caractères
  Ou un ensemble de chaînes de caractères
- Mémoriser de longues chaînes, tels les "pathnames"
- Trois types
  - Variables d'environnement
  - Variables locales
  - Variables arithmétiques
- Les variables d'environnement sont passées aux sous-shells
  Scripts, éditeurs...
- Variables locales : chaînes de caractères
- Variables arithmétiques : entiers, opérateurs arithmétiques...

Variables locales en C shell

`set varname [= string]`

- Définition
- Initialisation
- Exemples :

```
% set noclobber
% set cf = /home/enseign/marquet/C/gets
```

- Attention aux espaces :

```
% set title = "Disk usage summary report"
```

- Liste de mots ; utilisation des parenthèses

```
% set maillist = (marquet duchmol albert)
```


`$varname [[index]][:modifier]`

- Utilisation
- Expansion des variables par le shell
- Exemples

```
% ls -l $cf
% echo $title > disk.sum
% mail $maillist < urgent.txt
```

- Indexations des listes de mots

```
% mail $maillist[2] < precisions.txt
% mail $maillist[2-3] < all.txt
```

- Nombre de mots d'une liste

```
% echo Il y a $#maillist personnes dans le projet
Il y a 3 personnes dans le projet
```

- Une variable est-elle définie ?

```
% test $?maillist && mail $maillist < memo.txt
```

- Isoler un nom de variable

```
% ls -l $cf2
cf2 : Undefined variable
% ls -l ${cf}2
```

- Spécificateurs de “pathnames”
  - Les variables contiennent souvent des “pathnames”
  - Accès aux différents éléments du “pathname”

	Commande	Résultat
	`echo $cf`	`/home/marquet/gets.c`
racine	`echo $cf:r`	`/home/marquet/gets`
tête (*header*)	`echo $cf:h`	`/home/marquet`
queue (*tail*)	`echo $cf:t`	`gets.c`
extension	`echo $cf:e`	`.c`

- Destruction d’une variable

```
% unset cf
```

- Liste des variables locales : `set`

```
% set
argv    ()
cwd     /home/enseign/marquet
filec
history 32
home    /home/enseign/marquet
maillist   (marquet duchmol albert)
path    (. /bin /usr/bin  /home/enseign/marquet/bin)
prompt  %
shell   /bin/csh
status  0
term    vt300
user    marquet
```

Variables d'environnement

- Variables "globales"
- Variables passées aux sous-shells, commandes
- Convention : majuscule
- Stockage d'informations sur l'environnement de l'utilisateur
  Initialisées par le shell
  - Répertoire de login (`HOME`)
  - Nom d'utilisateur (`USER`)
  - ...
- Configuration du shell
  - Type du terminal (`TERM`)
  - Recherche des commandes (`PATH`)

```
% echo $PATH
.:/bin:/usr/bin:/home/enseign/marquet/bin
```

  (Cohérence avec la variable C shell `path`)

- Configuration de commandes

```
% echo $OPENWINDOWHOME
/usr/local/openwin
% echo $EMACSLOADPATH
/usr/local/emacs/lisp
```

`setenv VARNAME string`

- Définition et affectation
- Exemple

```
% setenv TERM xterm
```

`$VARNAME`

- Référence
- Exemple

```
% setenv PATH $PATH:/usr/local/bin
```

`printenv`
`setenv`
`env`

- Affiche la liste des variables d'environnement définies

```
% printenv
HOME=/home/enseign/marquet
SHELL=/bin/csh
TERM=xterm
PWD=/home/enseign/marquet/tmp
PATH=.:/bin:/usr/bin:/home/enseign/marquet/bin:
  /usr/local/bin
OPENWINHOME=/usr/local/openwin
EMACSLOADPATH=/usr/local/emacs/lisp
PRINTER=lexmarkv
```


Les substitutions

Exécution par le shell des substitutions :

- Expressions régulières pour les fichiers (`*.c`)
- Substitution de variables (`$HOME`)

(• Alias — `csh` et `tcsh`)

- Substitutions de commandes (‘ ‘)
- Protections \, " ", et ’ ’

Substitution de commandes

- Commandes qui affichent leur résultat

```
% date
Mon Jan 17 15:22:06 MET 1994
```

- Récupérer cette valeur affichée

  Utilisation des “back-quotes” — accent grave

```
% set d = `date`
```

  Équivalent à

```
% set d = (Mon Jan 17 15:22:06 MET 1994)
```

- Manipulation de ce résultat

```
% echo $d
Mon Jan 17 15:22:06 MET 1994
% echo Nous sommes le $d[3] $d[2]
Nous sommes le 17 Jan
% echo Il est $d[4]
Il est 15:22:06
```


- Utilisation directe

```
% echo "Il y a `who | wc -l` utilisateurs connectes"
Il y a 6 utilisateurs connectes
```

- Conversion des fins de ligne

```
% cat ~/.maillist
marquet
duchmol
albert
...
% mail `cat ~/.maillist` < memo.txt
```

- Exemple : éditer les fichiers contenant le mot Unix

```
% grep -l Unix *
philo.tex
scripts.tex
shell.tex
% emacs `grep -l Unix *`
```

Méta-caractères

- Caractères spéciaux utilisés par le shell

Symbole	Signification
“espace” “tabulation”	séparateurs d’arguments
“return”	terminaison de ligne
$	accès aux variables
* [ ] ? { } ~	expansion des noms de fichiers
< > & !	redirection
! ^	historique
|	pipe
;	délimiteur de commandes
()	groupement de commandes
\ ’ "	caractères d’échappement
‘	substitution de commandes
&	tâches de fond

- Échappement d’un seul caractère
  utilisation du \
- Exemples :

```
% echo $wf
/home/enseign/marquet/utils
% echo $wf est stocke dans \$wf
/home/enseign/marquet/utils est stocke dans $wf

% grep Oct\ 91 agenda.txt
...

% cp /home/enseign/marquet/utils/src/back/main.c \
/usr/backup
```

- De nombreux caractères à “échapper”
  Utilisation des quotes ' et "
- Les quotes " évitent toute substitution, sauf
  - Variables : $
  - Substitution de commandes : ‘‘

```
% echo "***** repertoire $wf ****"
**** repertoire /home/enseign/marquet/utils ****
% echo "***** ‘date‘ *****"
***** Mon Jan 17 15:22:06 MET 1994 *****
```

- Les quotes '' évitent toute substitution

```
% echo \
'Utilise la variable $USER dans ton programme' \
| mail duchmol
```


Here documents

- Redirection de l'entrée d'une commande
- Exemple

```
% set d = /usr/local/src
% set f = $d/prgm.c
% mail licence
Le programme C du prochain TP est disponible
dans le repertoire $d, il se nomme $f:t
^D
```

Les $d et $f:t ne sont pas expansés !

- Une solution

```
% cat > mail-it
Le programme C du prochain TP est disponible
dans le repertoire $d, il se nomme $f:t
^D
% mail licence < mail-it
% rm mail-it
```

- Gestion du fichier temporaire par le shell :

```
% mail licence << FINI
Le programme C du prochain TP est disponible
dans le repertoire $d, il se nomme $f:t
FINI
```

```
command [argument]...  << word
 command input
  ...
word
```


Aspects interactifs du C shell

- Historique des commandes
- Alias
- Accès aux noms de fichiers
- Navigation dans les répertoires
- Manipulation des jobs

Comment le C shell retrouve une commande

Pour exécuter interactivement une commande, le C shell

0. Réalise les substitutions :
   1. Expressions régulières pour les fichiers (`*.c`)
   2. Substitution de variables (`$HOME`)
   3. Substitutions de commandes (` ` `)
1. Recherche un alias

   Si oui

   – Réalise la substitution

   – Retourne en 1
2. Recherche si la commande est une commande *interne* au shell

   Exemple : `set`, `alias`, `cd`...

   L'exécute
3. Sinon, recherche un fichier

   Dans les *chemin de recherches*

   Variables `PATH`, `path`

   Table de hashage

   Commande `rehash`

COMMANDES ET FILTRES

Contents

Commandes et filtres

Philosophie Unix

- Commandes de base
- Combinaison de ces commandes de base
- Redirection des entrées/sorties, pipes
- Commande ≡ filtre

Redirection

	C shell	Bourne shell
entrée standard	<	<
entrée standard direct	<<	<<
sortie standard	>	>
sortie standard concaténée	>>	>>
sortie erreur		2>
		2>>
sortie standard + erreur	>&	
	>>&	
pipe	|	|

Filtres simples

`cat [-] [-bnv..] [filename]...`

- Affiche le contenu des fichiers
- Par défaut : l'entrée standard
- Concatène les fichiers
- Options
  - `-b` ou `-n` Numérotation des lignes
  - `-v` Filtre et affiche les caractères non imprimables
    Par exemple : `^c` (Control-C)...
- Exemples

```
% cat file1 file2 > file3
% ( cat file ; macommande) > result
```

```
more [options] [filename]...
less [options] [filename]...
```

- Affiche en mode page/page les fichiers
- Affiche l'entrée standard par défaut
- Exemple

```
% cat -v file | more
% ls -lag | more
% more this_enormous_file
```


`tee filename`

- Recopie de l'entrée standard
  - Sur la sortie standard, et
  - Dans `filename`
- Exemple

```
% ls | tee /tmp/foo | wc -l
2
% cat /tmp/foo
file1
file2
%
```

Des filtres, encore des filtres...

`tr [-cds] [s1 [s2]]`

- Copie `stdin` sur `stdout`
- Substitution des caractères `s1` par `s2`
- Intervalles : `a-z`, `0-9`, `C-G`
- Répétition : `a*5`, `=*0` ($0 \equiv \infty$)
- Caractères spéciaux : code ASCII de NEWLINE = `'\012'`
- Exemple, mise en minuscule

```
% cat > essai
Voici une phrase de ... SIX (6) mots
% tr A-Z a-z < essai
voici une phrase de ... six (6) mots
```


- Options
  - `-c` Complément de `s1` (dans l'ensemble des caractères ASCII)
  - `-d` Efface dans l'entrée les caractères de `s1`
  - `-s` Toute séquence dans `s1` est remplacée par un caractère unique dans `s2`
- Exemple, mise en mots

```
% tr A-Z a-z < essai | tr -sc a-z '\012'
voici
une
phrase
de
six
mots
```

```
cut [-clist] [filename]...
cut [-flist [-dc] [-s]] [-clist] [filename]...
```

- Couper les colonnes de chaque ligne d'un fichier
- Notion de champs, délimiteur
- Options
  - `-clist` Position des caractères à garder
    Liste croissante

`1,4,7`	colonnes 1, 4, et 7
`1-3,8`	colonnes 1 à 3, et 8
`-5,10`	colonnes (1) à 5, et 10
`3-`	colonnes 3 à (fin)

- Exemple, supprimer les deux premières colonnes

```
% cat file
* duchmol
$ marquet
^ albert
% cut -c3- file
duchmol
marquet
albert
```


- Options
  - `-flist` Liste des champs à garder
    Exemples : `-f1,3` ou `-f1-3,5`
  - `-dc` Délimiteur de champs *c* (défaut tabulation & espace)
    Exemple : `-d" "`
  - `-s` Suppression des lignes sans délimiteur
- Exemple

```
homel% cat passwd
duchmol:641qXHzHthhjk:7713:77:Pierre Duchmol:/home/licence/duchmol:/bin/c
marquet:UCRtTDCmnJEJM:7333:73:Philippe Marquet:/home/enseign/marquet:/bin
albert:ruFx8ojbEQgMM:7507:75:Francis Albert:/home/maitrise/albert:/bin/cs
homel% cut -d: -f1,5 < passwd
duchmol:Pierre Duchmol
marquet:Philippe Marquet
albert:Francis Albert
homel% cut -d: -f3-5 < passwd
7713:77:Pierre Duchmol
7333:73:Philippe Marquet
7507:75:Francis Albert
```

`paste [-dlist] file1 file2`

- Remise en forme des champs de 2 fichiers
- `-d` Définit les séparateurs de champs
- Exemple

```
% cat prenoms
Ale
Max
Sam
% cat noms
Lewis
Kunz
Schiper
% paste -d_ prenoms noms
Ale_Lewis
Max_Kunz
Sam_Schiper
```


- Autre exemple

```
% cat name
Sue
John
Helen
Peter
% cat number
167
876
465
7834
% cat city
LA
NY
SF
% paste name number city
Sue     167     LA
John    876     NY
Helen   465     SF
Peter   7834
```

- Il existe d'autres possibilités :

```
man 1 paste
```

`uniq [-cdu] [inputfile [outputfile]]`

- Détruit les lignes adjacentes dupliquées, ou
- Identifie ces lignes
- Options
  - `-c` Compte les lignes identiques
  - `-d` Ne garde que les lignes dupliquées et en un exemplaire
  - `-u` Ne garde que les lignes uniques dans `inputfile`

```
$ cat test
on se repete
on se repete
on se repete plus !
on se repete plus !
on se repete plus !
fin
$ uniq < test
on se repete
on se repete plus !
fin
$ uniq -u < test
fin
$ uniq -c < test
2 on se repete
3 on se repete plus !
1 fin
```


`sort [-rdfn...]  [-tc] [sort-field] [filename]...`

- Tri des lignes d'un fichier
  Ordre = code ASCII
- Options
  - `-n` Ordre numérique
  - `-d` Ordre du dictionnaire (seulement lettres, chiffres, espace)
  - `-r` Ordre inverse
  - `-f` Ignore les majuscules
- Exemple

```
homel% cat name
Sue
John
Helen
Peter
homel% sort name
Helen
John
Peter
Sue
```

- Exemple `-n`

```
homel% cat number
167
876
465
7834
homel% sort number
167
465
7834
876
homel% sort -n number
167
465
876
7834
```


- Exemple `-f`

```
homel% cat name
Sue
john
Helen
Peter
homel% sort name
Helen
Peter
Sue
john
homel% sort -f name
Helen
john
Peter
Sue
```

- Exemple -d

```
homel% cat name
Sue
john
Helen
P.eter
Paul
homel% sort name
Helen
P.eter
Paul
Sue
john
homel% sort -d name
Helen
Paul
P.eter
Sue
john
```


- Notion de champs

  sort-field : +*no* -*no* [flags]

- Options
  - -t*c* Délimiteur de champs
  - +*no* Numéro du champs de départ (≥ 0)
  - -*no* Numéro du champs d'arrêt (non inclus)
  - flags : rdfn...
  - Exemple : +0 -1rn
- Exemple

```
homel% cat passwd
duchmol:641qXHzHthhjk:7713:77:Pierre Duchmol:/home/licence/duchmol:/bin/c
marquet:UCRtTDCmnJEJM:7333:73:Philippe Marquet:/home/enseign/marquet:/bin
albert:ruFx8ojbEQgMM:7507:75:Francis Albert:/home/maitrise/albert:/bin/cs
homel% sort -t: +2 -3 passwd
marquet:UCRtTDCmnJEJM:7333:73:Philippe Marquet:/home/enseign/marquet:/bin
albert:ruFx8ojbEQgMM:7507:75:Francis Albert:/home/maitrise/albert:/bin/cs
duchmol:641qXHzHthhjk:7713:77:Pierre Duchmol:/home/licence/duchmol:/bin/c
homel% sort -t: +4 -5 passwd
albert:ruFx8ojbEQgMM:7507:75:Francis Albert:/home/maitrise/albert:/bin/cs
marquet:UCRtTDCmnJEJM:7333:73:Philippe Marquet:/home/enseign/marquet:/bin
duchmol:641qXHzHthhjk:7713:77:Pierre Duchmol:/home/licence/duchmol:/bin/c
```

`comm [-123] filename1 filename2`

- Sélection de lignes entre 2 fichiers (triés)
- 3 types de lignes :
  1 Lignes de `filename1` ($\notin$ `filename2`)
  2 Lignes de `filename2` ($\notin$ `filename1`)
  3 Lignes communes
- Paramètres : désignent les lignes à supprimer
- Exemple, les nouvelles lignes de `file` :

```
% comm -13 file.old file
```

- Exemple, les lignes supprimées de `file` :

```
% comm -23 file.old file
```


`diff [-c...]  filename1 filename2`

- Comparaison des deux fichiers
- Assez “intelligent”
- `-c` produit un fichier exploitable par `patch`

- Exemple

```
homel% cat > afile
A
B
C
D
E
F
G
homel% cat > anotherfile
A
B
C
d
E
F
g
h
homel% diff afile anotherfile
4c4
< D
---
> d
7c7,8
< G
---
> g
> h
```


`cmp [-s] filename1 filename2`

- Comparaison par octets
- Filename = `-` ⇒ `stdin`
- `-s` Silencieux
  Code de retour 1 si différence
- Exemple

```
homel% cmp afile anotherfile
afile anotherfile differ: char 7, line 4
homel% cmp afile thisfile
```

`join [-j[1|2]]` *m* `[-t`*c*`] [-o list] file1 file2`

- Opérateur de jointure
- Jointure sur des champs triés
- Options :
  - `-t`*c* : Séparateur de champs
  - `-j[1|2]` *m* : Champs de la jointure
    `-j1` : Premier fichier
    `-j2` : Second fichier
    Numérotation des champs à partir de 1
  - `-o` *list* champs en sortie
    Éléments de *list* de la forme *m.n*
    m numéro du fichier
    n numéro du champs

- Exemple

```
homel% cat passwd
duchmol:641qXHzHthhjk:7713:77:Pierre Duchmol:/home/licence/duchmol:/
marquet:UCRtTDCmnJEJM:7333:73:Philippe Marquet:/home/enseign/marquet
albert:ruFx8ojbEQgMM:7507:75:Francis Albert:/home/maitrise/albert:/b
podvin:Myg7Thg6gTrdd:7509:75:Maurice Podvin:/home/maitrise/podvin:/b
homel% sort -t: +2 -3 passwd
marquet:UCRtTDCmnJEJM:7333:73:Philippe Marquet:/home/enseign/marquet
albert:ruFx8ojbEQgMM:7507:75:Francis Albert:/home/maitrise/albert:/b
podvin:Myg7Thg6gTrdd:7509:75:Maurice Podvin:/home/maitrise/podvin:/b
duchmol:641qXHzHthhjk:7713:77:Pierre Duchmol:/home/licence/duchmol:/
homel% sort -t: +2 -3 passwd > passwd.sort
homel% cat group
enseign:73
maitrise:75
licence:77
miage:80
homel% join -t: -j1 4 -j2 2 -o 1.1 2.1 passwd.sort group
marquet:enseign
albert:maitrise
podvin:maitrise
duchmol:licence
```

`look [-df] [-tc] string [file]`

- Recherche des lignes égales à (commencant par) `string`
- Dans un fichier trié
  - Ordre de `sort`
  - `-d` ordre du dictionnaire
  - `-f` majuscules et minuscules indifférentes
- `-t`*c* tous les caractères à la droite de *c* sont ignorés

Base de données relationnelle

- Commandes opérateurs de base de données
- `sort cut join grep look`
- Pas forcément efficace !

Expressions régulières

- 3 spécifications différentes :
  - Shell
  - Commandes
  - `sed` "*stream editor*"
- Manière synthétique de représenter un ensemble de chaînes de caractères
  - Chaînes normales
  - Méta-caractères

Syntaxe shells

- Noms de fichiers
- Substitués par l'interpréteur

syntaxe	*sens*
`abc`	cette chaîne
`?`	un caractère quelconque
`*`	des caractères
`[xyz]`	`x` ou `y` ou `z`
`[a-g]`	un caractère entre `a` et `g`
`[!xyz]`	un caractères sauf `x`, `y` ou `z`

Exemple de la syntaxe shells

```
% ls
File1    File2    File3
fichier  prog1.c  prog2.c

% echo File?
File1 File2 File3

% echo ?i*
File1 File2 File3 fichier

% echo *.c
prog1.c prog2.c

% echo *2*
File2 prog2.c

% echo [!Ff]*
prog1.c prog2.c
```


Recherche de fichiers

`find dir expr command`

- Recherche dans une arborescence
- Des fichiers satisfaisant `expr`
- Application de `command`
- `dir` racine de l'arborescence
- `expr`
  - `( ) -o -a` opérateurs logiques
  - `name` *`reg-expr-shell`*
  - `-user` *`name`*
  - `-size` *n*
    `-size +`*n* taille $n \times 512$ octets
    `-size -`*n*
  - `-atime` *n*
  - Autre possibilitées : `man 1 find`
- `command`
  - `-print` affiche le nom du fichier
  - `-exec` *`unix-cmd`*
    Le nom du fichier : `{}`

- Exemple : les fichiers .c de plus de 40 Ko

```
% find . -name '*.c' -size +80 -print
./pvm3/src/lpvm.c
./pvm3/src/pvmd.c
```

- Exemple : mes fichiers

```
% find / -user $USER -print
```

- Exemple : nettoyage

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


Expressions régulières de commandes

- Expressions régulières par ligne
- Commandes `grep` `sed` `awk` `expr`

syntaxe	*sens*
c	caractère ordinaire
`.` (dot)	un caractère quelconque
`[xyz]`	`x` ou `y` ou `z`
`[A-G]`	interval
`[^xyz]`	sauf `x`, `y` ou `z`
`^`	début de ligne
`$`	fin de ligne
`*`	opérateur de répétition (≥ 0)
`\{`*m*`,`*n*`\}`	répétions entre *m* et *n* fois
`\`	échappement

Exemples d'expressions régulières

- `^$` : lignes vides
- `mot` : lignes contenant `mot`
- `^[A-Z]`
- `[a-z][a-z]$`
- `[ABCD]\{2,10\}$`
- `[^:][^:]*:*$`

Recherche de motifs dans un fichier grep

`grep [-vin] regexpr [file]...`

- Cherche et affiche les lignes contenant `regexpr`
- Options
  - `-v` : complémentaire
  - `-i` : maj. et min. indifférentes
  - `-n` : affiche les numéros de lignes

```
$ ps aux | grep "^marquet"
marquet  14551  ... 0:10 ghostview poly.ps
marquet  15818  ... 0:02 gs -dQUIET -dNOPAUSE -
marquet  13883  ... 0:05 -tcsh (tcsh)
```

Calcul d'expressions `expr`

`expr argument...`

- Calcule la valeur d'une expression
- Expressions booléennes : `=` `!=` `>` `<` `>=` `<=`

  `expr 45 != 20`
- Expresions arithmétiques : `+ - * / %`

  `expr 2 + 1`
- Opérateur « : » : `expr string : reg-expr`

  Renvoie le nombre de caractères reconnus

  (0 si non correspondance)

- Exemple : incrémentation de la variable a

  ```
  a=`expr $a + 1`
  ```
- Exemple : nombres de caractères de `$VAR`

  ```
  expr $VAR : '.*'
  ```
- Exemple : le répertoire courant est-il sous `/home/`

  ```
  jesuisla=`pwd`
  lghome=`expr /home/ : '.*'`
  lgpwd=`expr $jesuisla : '/home/.*'`
  expr $lghome \<= $lgpwd
  ```

Editeur sed

`sed [-n] [-e script ] [file]...`

- “*stream editor*”
- Langage de manipulation de chaînes de caractères
- Manipulation ligne par ligne (*pattern space*)
- Commandes exécutées sur chaque ligne
  `a` append, `d` delete, `p` print, `s` substitute...
- Application contextuelle des commandes
  Numéros de lignes
  Dernière ligne (`$`)
  Expression régulière
- Option `-n` : no output

Règles de `sed`

`[address [,address]] function [arg]...`

- `address` :
  - Numéro de ligne
  - `$` pour la dernière ligne
  - Expression régulière
- `address, address` : intervalle concerné
- `function` : une lettre `abcdDgGHilnNpPqrstwy`

Un premier exemple de sed

```
% cat test-sed
marquet 7333 Philippe Marquet
albert 7507 Francis Albert
podvin 7509 Maurice Podvin
duchmol 7713 Pierre Duchmol

% sed -n -e 2,3p < test-sed
albert 7507 Francis Albert
podvin 7509 Maurice Podvin

% sed -e s/Pierre/Peter/ < test-sed
marquet 7333 Philippe Marquet
albert 7507 Francis Albert
podvin 7509 Maurice Podvin
duchmol 7713 Peter Duchmol
```


Les commandes de sed

- `a\` : (append) ajoute du texte
- `c\` : (change) remplace la ligne
- `i\` : (insert)
- `d` : (delete)
- `w` *`file`* : écrit la ligne dans file
- `p` : (print) envoie sur stdout
- `n` : ligne suivante (`-f`)
- `q` : (quit)
- `y/str1/str2/` : (yank), même longueur

Fichier de commandes sed

- Option -f file :
  file contient des commandes de sed

```
% cat prog.sed
1a\
albert 7507 Francis Albert
% cat test-sed
marquet 7333 Philippe Marquet
podvin 7509 Maurice Podvin
duchmol 7713 Pierre Duchmol
% sed -f prog.sed < test-sed
marquet 7333 Philippe Marquet
albert 7507 Francis Albert
podvin 7509 Maurice Podvin
duchmol 7713 Pierre Duchmol
```


Les commandes de sed

- b *label* et : *label* : branch !
- ! : not, complémentaire des adresses
- Hold space : une ligne sous le coude !
- Commandes :
  - g hold → pattern
  - G idem mais ajoute
  - h pattern → hold
  - H idem mais ajoute

Exemple hold space

- Afficher la dernière ligne commençant par X

```
% cat > X.sed
/^X/h
$g
$p

% cat > f
X premiere
Attention...
X seconde
Fin

% sed -n -f X.sed < f
X seconde

% cat >> f
X troisieme

% sed -n -f X.sed < f
X troisieme
```


$\boxed{\texttt{s/}\textit{reg-expr}\texttt{/}\textit{replacement}\texttt{/}\textit{flags}}$

- `s` : (subsitute) commande la plus importante...
- Extension expression régulière en `sed` :
  - Stockage `\(`*reg-expr*`\)`
  - Rappel : `\1 \2 \3`
- Flags :
  - `p` : print
  - `g` : global (plusieurs substitution par ligne)
  - $n \in [1 - 512]$: occurrence à substituer
  - `w` *file* : si oui écrire dans le fichier

- Exemple sed s : options de substitution

```
% cat > f
aaa bbb aaa bbb aaa
aaa ccc
% sed -e 's/aaa/AAA/' < f
AAA bbb aaa bbb aaa
AAA ccc
% sed -e 's/aaa/AAA/g' < f
AAA bbb AAA bbb AAA
AAA ccc
% sed -e 's/aaa/AAA/g3' < f
aaa bbb aaa bbb AAA
aaa ccc
```

- Exemple sed s : inversion des champs 1 et 2

```
% cat > f
marquet:philippe:7713
duchmol:pierre:7333
albert::francis
% sed -e 's/^\([^:]*\):\([^:]*\)/\2:\1/' < f
philippe:marquet:7713
pierre:duchmol:7333
:albert:francis
% echo '\$[] !' | sed -e 's/\\\$\[]/ouff/'
ouff !
```


Grand est sed

- `head`

  ≡ `sed -n -e 1,10p` ≡ `sed -e 10q`

- `cut -d: -f1`

  ≡ `sed -n -e 's/^\([^:]*\):.*$/\1/p'`

- etc.

Il y a mieux (pire) : `awk`

Conclusion sur les commandes et filtres

- Beaucoup de commandes...
- Savoir qu'elles existent
- Savoir ce que l'on peut en attendre
- `man`

Exemple

Un vérificateur orthographique

- Commande `spell`

  Vérifie que les mots du fichier passé en arguments (`stdin` par défaut) sont dans le dictionnaire

  Options

  – `-d`*`langue`* Dictionnaire de la *`langue`*
- Fichiers :
  – `$FILE.tex` : le source TEX ou LATEX
  – `$HOME/.dicoperso` : les mots corrects qui ne sont pas dans le dictionnaire
  – `$FILE.fot` : les mots corrects du document qui ne sont pas dans le dictionnaire
  – Les `.fot` :
  Un mot par ligne
  Trié (`sort`)
- Commentaires TEX et LATEX
  – De % à fin de ligne
  – Sauf \%
- But : afficher les mots incorrects du source LATEX

- Algorithme :
  1. Supprimer les commentaires du source LaTeX
  2. Identifier les mots du source $\notin$ `spell`
  3. Les trier
  4. Enlever les mots $\in$ `.dicoperso`
  5. Enlever les mots $\in$ `.fot`
  6. Afficher les mots restant

- Supprimer les commentaires

```
homel% cat file.tex
un
deux % et de 2
trois \% quate
cinc
% et de 5
six
```

- Supprimer les commentaires (i.e %* sauf \%*)

```
homel%  sed -e 's/\([^\]\)%.*$/\1/' \
        < file.tex > file.tex.1
homel% cat file.tex.1
un
deux
trois \% quate
cinc
% et de 5
six
```

- Supprimer les commentaires en début de ligne :

```
homel% grep -v "^%" < file.tex.1 > file.tex.2
homel% cat file.tex.2
un
deux
trois \% quate
cinc
six
```


- Identifier les mots du source $\notin$ `spell`

```
homel% spell -dfrancais < file.tex.2 > file.tex.3
homel% cat file.tex.3
quate
cinc
```

- Trier ces mots

```
homel% sort < file.tex.3 > file.tex.4
```

- Supprimer les mots $\in$ `file.fot` ou `$HOME/.dicoperso`

```
homel% cat file.fot $HOME/.dicoperso
...
quate
...
homel% comm -23 file.tex.4 file.fot \
       > file.tex.5
homel% comm -23 file.tex.5 $HOME/.dicoperso \
       > file.tex.6
```

- Afficher les mots restants

```
homel% cat file.tex.6
cinc
```

SHELL SCRIPTS (BOURNE SHELL)

Contents

Shell-scripts

- Shell

  (– Interface utilisateur pour lancer des commandes)
  - Exécution de scripts
- Shell-script
  - Utilisation non-interactive d'un shell
  - Texte des commandes dans un fichier
  - Fichier appelé comme une commande
  - Spécification de l'interpréteur : ligne 1

    ```
    #! /bin/csh
    #! /bin/sh
    ```

    Mode debug

    ```
    #! /bin/sh -vx
    ```

    Détection des variables non définies

    ```
    #! /bin/sh -u
    ```

    Commentaires après le #
- On choisit le Bourne shell (`/bin/sh`)

Exemple de shell-script

- Le fichier de script

```
% cat /tmp/informations
#! /bin/sh

# Introduction
clear ; echo "Bonjour"

# Quelques informations
echo "Tu es $USER sur $HOST"
echo "en `pwd`"
```

- Le droit d'exécution sur ce fichier

```
% ls -l /tmp/informations
-rwxr-x--- 1 marquet ... informations
```

- Exécution du script

```
% /tmp/informations
Boujour
Tu es marquet sur homel
en /tmp
```


Variables de sh

- `$HOME` : répertoire de base
- `$PATH` : répertoires de recherches des commandes
- `$PS1`, `$PS2` : prompts

```
$ echo $PS1
$
$ PS1="$USER@$HOST >"
marquet@homel >
```

- `$#` : nombre de paramètres du script
- `$0` à `$9` : paramètres 0 (nom de la commande) à 9
- `$*` : tous les paramètres
- `$@` : tous les paramètres
- `$?` : code de retour (erreur) de la dernière commande
- `$$` : numéro de processus du shell courant
- `$!` : numéro de processus de la dernière commande background (`&`)
- `$-` : flags/options du shell

Exemple

- Un shell script

```
% cat petit-shell-script
#! /bin/sh -eu

echo "  commande $0"
echo "    avec $# arguments"
echo "    processus $$"
echo "  param 1 : $1"
echo "  les options du shell sont $-"
echo "  $*"
```

- Son exécution

```
% petit-shell-script hello world
  commande ./petit-shell-script
    avec 2 arguments
    processus 20830
  param 1 : hello
  les options du shell sont eu
  hello world
```


Accès aux variables en sh

- `${varname}` : la valeur de la variable
- `${varname:-word}`

  `${varname:=word}` : si la variable n'existe pas ou est nulle, l'affecte de `word` ; retourne la valeur de la variable
- `${varname:?message}` : affiche un message et termine le script (ou la fonction) si la variable n'est pas définie ou de valeur nulle

  Le message par défaut (forme `${varname:?}`)

  `varname: parameter null or not set`
- `${varname:+word}` : si la variable existe et est non nulle l'affecte de la valeur `word`, sinon de la chaîne vide

Évaluation de `word`/`message` seulement si nécessaire (“`andthen`”, “`orelse`”)

Exemple

```
% cat vartest
#! /bin/sh
test=$4
echo ${test:=alpha}
echo ${1:-beta}
echo ${2:+gamma}
echo ${3:?"parametre absent"}

% vartest
alpha
beta

vartest: 3: parametre absent

% vartest un deux trois quatre
quatre
un
gamma
trois
```


Entrées-sorties d'un shell-script

- `echo args` : affiche ses arguments sur `stdout`
- `echo args 1>&2` : affiche sur `stderr`
- `read` *`names`* :
  Lit une ligne de stdin
  Associe les champs aux *`names`*
  Champs séparés par `$IFS`
- Valeur par défaut de `$IFS` : NEWLINE, TAB, SPACE

Exemple d'entrée-sortie

```
% more filtre
#! /bin/sh

echo "  vers sortie standard (stdout)"
echo "  vers erreur standard (stderr)" 1>&2

IFS=:

read FLD1 FLD2
echo "  entree (stdin): $FLD1 $FLD2"

% echo "Peter:Pan:Field 3" | filtre > result
  vers erreur standard (stderr)

% more result
  vers sortie standard (stdout)
  entree (stdin): Peter Pan:Field 3
%
```


Structures de contrôle

Le shell est un langage de programmation :

- Séquence : `command ; command`
- Sous-shell : `( ... )`
- Boucle : `for ... do ... done`
- Condition : `if ... then ... else ... fi`
- Tant que : `while ... do ... done`
- Branchement : `case ... esac`
- Fonctions : `name() { ... }`

Fonctions shell

```
la-fonction ()
 {   ...
 }
```

- Définition de fonctions
- Appel comme une commande shell

  `la-fonction arg1 arg2`

- Accès aux paramètres par $1 à $9
- Retour de résultat
  - `return [valeur numérique]`
    accès par `$?`
    ou utilisation comme prédicat d'une structure de contrôle
  - affiche une chaîne de caractères
    accès par `` `...` ``
- Peut redéfinir les commandes Unix
  Mais pas les commandes spéciales du shell :
  `echo exit export pwd read test ...`

Exemple de fonctions

```
% cat test-funct
#! /bin/sh

arguments()
{
  echo "  fonction arguments : $*"
}

echo "  dans le shell : $*"
arguments "hello world"
%
% test-funct coucou
  dans le shell : coucou
  fonction arguments : hello world
%
```

Boucle `for`

`for` *var* `in` *word-list* `do` *commands* `done`

- *word-list* : liste de mots séparés par des TAB et SPACE
- Exécution des *commands* pour chacun des mots
- Valeur de `$`*var* lors d'une itération : le mot

Exemple de `for`

```
% cat test-for
#! /bin/sh
for i in Lenine Staline Kroutchev Brejnev
do
  echo "  $i fut maitre de la Russie !"
done
echo "  -"
# aussi possible : for i in *
for i in `ls *`
do
  echo "  $i est un fichier"
done
% test-for
  Lenine fut maitre de la Russie !
  Staline fut maitre de la Russie !
  Kroutchev fut maitre de la Russie !
  Brejnev fut maitre de la Russie !
  -
  filtre est un fichier
  test-for est un fichier
```

Différence entre $* et $@

- Dans un shell script, $* et $@ référencent tous les arguments
- Exemple

```
homel% cat all1
#! /bin/sh
for i in $*
do
  echo $i
done

homel% all1 a b c
a
b
c
homel% all1 'a b' c
a
b
c
```

```
homel% cat all2
#! /bin/sh
for i in $@
do
  echo $i
done

homel% all2 a b c
a
b
c
homel% all2 'a b' c
a
b
c
```

- Les arguments sont réexaminés au moment de la boucle
- L'espace dans le premier argument crée deux arguments !

- Protection par " des $* et $@

```
homel% cat all3
#! /bin/sh
for i in "$*"
do
  echo $i
done

homel% all3 a b c
a b c


homel% all3 'a b' c
a b c
```

```
homel% cat all4
#! /bin/sh
  for i in "$@"
do
  echo $i
done

homel% all4 a b c
a
b
c
homel% all4 'a b' c
a b
c
```

Règles :

- $@ ≡ $* : les arguments de la commande sont réexaminés
- "$*" est un seul mot composé de tous les arguments
- "$@" fournit exactement les arguments

Alternative `if .. then ... else`

`if list then list [else list] fi`

- Exécution de la commande condition *list*
- Code de retour de la commande
- Vrai si nul ($\neq$ langage C)
- Exécution d'une des deux listes de commandes

`[elif list then list]`

- Emboîtement des conditions

Itération `while`

`while list do list done`

- Exécution de la commande condition *list*
- Code de retour de la commande
- Vrai si nul ($\neq$ langage C)
- Exécution de la liste de commandes tant que la condition est vraie

Lecture d'un fichier ligne par ligne

- Traiter séquentiellement une à une les lignes d'un fichier

Première solution

- Définition de la fonction `readline`
- `readline file i` écrit la ligne i du fichier `file`

```
readline () {
  fichier=$1
  ligne=$2
  head -$ligne $file | tail +1
}
```

- Utilisation pour traiter un fichier (`$file`) ligne par ligne

```
nlignes=`wc -l < $file`
i=1
while test $i -le $nlignes
do
  laligne=`readline $file $i`
  # traitement de la ligne
  ...
  i=`expr $i + 1`
done
```


La bonne méthode

- Utilisation d'une boucle `while` associée à `read`

```
cat $file | while read laligne
do
   # traitement de la ligne
   ...
done
```

ou

```
while read laligne
do
   # traitement de la ligne
   ...
done < $file
```

Condition `test`

- Commande `test`
- Utilisée dans les conditions en Bourne shell
- Évalue l'expression
- Retourne 0 si sa valeur est vraie
- Retourne $\neq 0$ sinon

```
test expr
[ expr ]
```

- Expression sur les fichiers
  - `-f` *`file`* : existe et non répertoire
  - `-d` *`file`* : existe et est un répertoire
  - `-s` *`file`* : existe et est de taille non nulle
  - `-r` *`file`* : existe et droit en lecture
  - `-w` *`file`* : existe et droit en écriture
  - `-x` *`file`* : existe et droit d'exécution

- Chaînes de caractères
  - *`str1`* `=` *`str2`*
    `str1` `!=` *`str2`*
    Vrai si les chaînes sont égales (différentes)
  - `-z` *`str`* : vrai si chaîne de longueur nulle
  - `-n` *`str`* : vrai si chaîne de longueur non nulle
- Nombres, comparaison
  - *`n1`* `-eq` *`n2`* égaux
    `n1` `-ne` *`n2`* non égaux
    `n1` `-gt` *`n2`* plus grand que
    `n1` `-ge` *`n2`* plus grand ou égal
    `n1` `-lt` *`n2`* plus petit que
    `n1` `-le` *`n2`* plus petit ou égal
- Opérateurs
  - `!` unaire de négation
  - `-a` et logique
  - `-o` ou logique
  - `(` *`expr`* `)` paranthésage
    `\(` et `\)` sinon interprétées par le shell

Exemple de test

```
% cat decompte
#! /bin/sh
var=10
while test $var -ge 4
do
  echo "  $var"
  var=`expr $var - 1`
done
% decompte
  10
  9
  8
  7
  6
  5
  4
```


Utilisation de test

Protection et vérification

```
% cat protect
#! /bin/sh

if test $# -ne 1
then
  echo "  erreur nombre d'arguments" 1>&2
  exit 1
fi

FILE=$1
if test ! -f $FILE
then
  echo "  erreur : $FILE n'existe pas" 1>&2
  exit 1
else
  echo "  $FILE existe, je l'ai rencontre "
fi
cat $FILE
%
% protect f1 f2
  erreur : nombre d'arguments
% protect xx
  erreur : xx n'existe pas
% protect filtre
  filtre existe, je l'ai rencontre
ligne 1
ligne 2
...
```

Constructeur case

```
case string in
label) ...  ;;
 ...
 *) ...  ;;
esac
```

- *string* est comparée aux `label`
- Exécute les commandes correspondant au premier label
- Les `label` sont des expressions régulières du shell

Exemple

```
case $file in
*.p)
    echo "$file is a pascal program" ;;
*.c)
    echo "$file is a C program" ;;
*.cc|*.c++)
    echo "$file is a C++ program" ;;
*)
    echo "I'm rather puzzled"
    echo "   with that $file " ;;
esac
```


Commandes spéciales

- Commandes internes au shell
- Commandes exécutées directement par le shell
  Leur code est dans le shell
- Liste de ces commandes :
  ($\star$ cf infra)
  - `:` ne fait rien et renvoie un code de retour 0
  - `break` sortie d'un `for` ou d'un `while`
  - `continue` sortie d'un `for` ou d'un `while`
  - `cd [arg]`
  - `echo [arg]`
  - `eval [arg]...` $\star$
  - `exec [arg]...` $\star$
  - `exit [no]` quitte le shell, positionne le status à `no`
  - `export [name]...` rend les variables `name` accessibles aux sous-shells
  - `hash`
  - `pwd`
  - `read [name]...`
  - `readonly [name]...`
  - `return [no]` sortie d'une fonction

- Liste des commandes (suite)
  - `set [--aefhkntuvx [arg]...]` ⋆
  - `shift` ⋆
  - `test`
  - `time [arg]` mesure du temps pour une commande
  - `trap [arg] [n]...` capture des interruptions logicielles
  - `type [name]...` ⋆
  - `ulimit [n]`
  - `umask [nnn]`
  - `unset [name]...`
  - `wait [n]`

Évaluation, évaluation... eval

`eval [arg]...`

- Exécution par le shell en deux phases :
  1. Évaluation des arguments de eval
  2. Réévaluation de ces arguments et exécution de la commande
- Exemple

```
% var= '$val'
% echo $var
$val
% val='Unix'
% eval echo $var
Unix
```

- Exemple : accès au dernier argument d'un script (≤ 9)

```
eval dernier=\$"$#"
echo $dernier
```

Exécution... exec

`exec [arg]`

- Provoque l'exécution de son argument
- Cet argument est considéré comme une commande
- Pas de retour possible au shell
- Exemple :

```
% exec rlogin rocs.lifl.fr
```


Positionner les variables set

`set [-aefhkntuvx] [argument]...`

- Permet de positionner des options du shell
- `set` sans options affiche les variables d'environnement
- La valeur de `argument` est affectées aux paramètres $0 à $9
- Exemple

```
% date
Thu Feb 24 13:34:12 MET 1994
% set `date`
% echo $3 $2
24 Feb
```

Décaler les paramètres shift

- Décalage des paramètres
- Accès aux paramètres > 9
- Perte de $0

  $0 ← $1

  $1 ← $2

  ...

  $9 ← paramètre 10
- Exemple

```
% cat test-shift
#! /bin/sh
echo "  $# params : $*"
echo "  premier : $1"
shift
echo "  $# params : $*"
echo "  premier : $1"

% test-shift Voici les parametres
  3 params : Voici les parametres
  premier : Voici
  2 params : les parametres
  premier : les
```


Type d'une commande type, which

`type commande [commande]...`

- Permet de connaître le chemin d'accès d'une commande
- Ou le type d'une commande
- Exemple

```
% type ls rm cp cd pwd mv
ls is /usr/bin/ls
rm is a function
rm(){
/bin/rm $*
}
cp is /usr/bin/cp
cd is a shell builtin
pwd is a shell builtin
mv is hashed (/usr/bin/mv)
```

`which commande [commande]...`

- Recherche des commandes dans $PATH
- Exemple

```
% which sh xfig
/bin/sh
/usr1/X11/bin/xfig
```

Interprétation d'une ligne de commandes

- Lecture jusque `;` ou `&`
- Variables remplacées par leur valeur
- Substitution des commandes et alias
- Découpage des arguments
- Traitement des caractères spéciaux
- Génération des noms de fichiers
- Redirections des entrées-sorties
- Recherche de la commande (built-in, `$PATH`)

Options de `sh`

- `-i` : interactif
- `-c command` : exécution de `command`
- `-e` : exit si erreur d'une commande
- `-f` : pas de génération des noms de fichiers
- `-n` : commandes non exécutées
- `-u` : variables non définies = erreurs
- `-v` : verbose
- `-x` : affiche les commandes comme exécutées
- Exemple : `sh -nv file` ≡ `cat file`

Autres shells `csh` et `tcsh`

- Mêmes fonctionnalités que `sh`
- Syntaxe à la C
- Expressions `( )`, `==`, `!=`, `&&`, `||`
- Contrôle

  ```
  if ( . ) then ... endif
  foreach var ( . ) ... end
  while ( . ) ... end
  switch ( . ) case . : ... break endsw
  ```
- Variables : `prompt`, `argv`, `status`...

Configuration des shells

- Fichiers invoqués par `sh` au login :

  `/etc/profile` : pour tous

  `$HOME/.profile` : personel
- Pour `csh` et `tcsh`

  `.cshrc`, `.tcshrc`

  `.login`, `.logout` : login shell

  `.history` : sauvegarde de l'historique

Exemple : fichiers archives

- Problème : envoyer une copie de tous les fichiers d'un repertoire par mail
- Solution :

```
% for i in *
> do
> echo "==== fichier $i ===="
> cat $i
> done | mail duchmol@ens.lifl.fr
```

- Du point de vue de `duchmol@ens.lifl.fr` : comment reconstituer les différents fichiers ?

- Idée : contruire un fichier archive auto-extractable
- Exemple :

```
% cat auto
#! /bin/sh
#
cat > toto << Fin_de_fichier
Premiere ligne de toto
Ligne 2 de toto
Fin_de_fichier
cat > titi << Fin_de_fichier
Ligne 1 de titi
Ligne 2 de titi
Ligne 3 de titi
Fin_de_fichier
```

Recréer les fichiers `toto` et `titi` :

```
% auto
```

- Création d'une telle archive
  - Commande `archiv`
  - Les fichiers à inclure sont passés en paramètre
  - Le fichier auto-extractable est produit sur la sortie standard

```
% cat archiv
#! /bin/sh

# entete du script produit
echo "#! /bin/sh"
echo "# Archive auto-extractable"

# Pour chacun des parametres
for i in "$@"
do
  echo "cat > $i << Fin_de_fichier"
  cat $i
  echo "Fin_de_fichier"
done
```


- Problème : si une ligne d'un des fichiers débute par `Fin_de_fichier`
- Solution :
  1. Lors de la création de l'archive
     Remplacer les débuts de ligne par `X_`
  2. Lors de la reconstruction des fichiers
     Remplacer les `X_` par des débuts de ligne

```
#! /bin/sh
#
ala_cat > toto << Fin_de_fichier
X_Premiere ligne de toto
X_Ligne 2 de toto
Fin_de_fichier
ala_cat > titi << Fin_de_fichier
X_Ligne 1 de titi
X_Ligne 2 de titi
X_Ligne 3 de titi
Fin_de_fichier
```

- Version 2 :

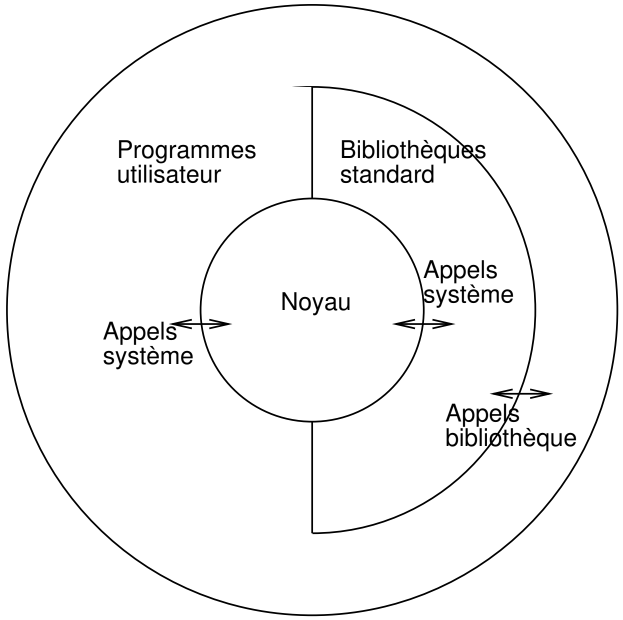
```
#! /bin/sh

# entete du script produit
echo "#! /bin/sh"
echo "# Archive auto-extractable"

# Pour chacun des parametres
for i in "$@"
do
  echo "sed -e 's/^X_//g' > $i << End_of_file"
  cat $i | sed -e 's/^/X_/g'
  echo "End_of_file"
done
```


INTERFACE DE PROGRAMMATION

Contents

Unix et le C

- Unix est principalement écrit en C
- Un interface naturelle avec le C
- Notion de noyau et de bibliothèque



- Appel système ou appel d'une fonction de la bibliothèque standard
  - Semblables
    Comme des appels de fonctions C
  - Différents
    - ▷ Appel système :
      Pas d'édition de liens
      Exécution d'un morceau de code du système
    - ▷ Bibliothèque standard
      Abstraction de plus haut niveau
      Nécessite une édition de liens avec la bibliothèque

Appels système

- Grandes catégories :
  - Manipulation du système de fichiers
  - Gestion des processus
    - ▷ Processus ≡ exécution d'un programme
    - ▷ Allocation de ressources pour ce processus est réalisée par le système : mémoire...
    - ▷ Lancement, arrêt et ordonnancement des processus contrôlés par le système
  - Communications entre processus
  - Informations

- Retour d'un appel système
  - Déroulement anormal ⇒ retourne -1
  - La variable externe `errno` indique le type de l'erreur

    ```
    extern int errno ;
    ```

  - Exemples de valeurs

1	Problème de permission
2	Fichier ou répertoire inexistant
3	Processus inexistant
5	Erreur d'entrée sortie
11	Table des processus pleine
...	...

  - Cf `/usr/include/sys/errno.h`
    Ou le manuel en ligne de la primitive
- Réalisation d'un appel système
  ⇒ Interruption du processus appelant

Bibliothèques standard

- Nombreuses bibliothèques standard
  Exemples :
  - Entrées-sorties formatées/bufferisées
  - Fonctions mathématiques
  - Allocation de mémoire dynamique
- Avantages
  - Abstraction de plus haut niveau
    Exemple : manipulation de chaînes de caractères
  - Performance :
    Nombre d'appels au moniteur réduit
    Exemple : allocation mémoire `malloc()/sbrk()`

Interface shell

- Écrire de nouvelles commandes
  Réécrire des commandes déjà existantes
- Appel de ces commandes depuis le shell sous la forme

```
nom [arguments]...
```

Accès aux arguments de la commande

- Paramètres de la fonction `main()`

```
main (int argn, char * argv[]) ;
```

`argn`	nombre d'arguments + 1
`argv[0]`	nom de la commande
`argv[1]` à `argv[argn]`	arguments

- Mémorisation du nom de la commande
  ⇒ Points d'entrée différents dans un même programme
  Exemple :

```
homel% ls -iF /usr/ucb/vi /usr/ucb/edit
5007 /usr/ucb/edit*    5007 /usr/ucb/vi*
```


- Exemple, ma commande echo

```
main (int n, char *v [])
{
  int i ;
  for (i=1 ; i<n ; i++)
    printf ("%s%c", v[i], (i<n-1) ? ' ': '\n') ;
}
```

Accès à l'environnement shell

- Variables d'environnement C-shell
  Variables exportées du Bourne shell
- Trois moyens

1. Variable externe `environ`

   `extern char **environ ;`

   Tableau de pointeurs sur des chaînes de la forme

   `nom = valeur`

   Exemple : la commande shell `printenv`

```
extern char **environ ;
main ()
{
  char **p, *q ;
  p = environ ;
  while (q = *p++)
    printf ("%s\n", q) ;
    }
```


2. Forme générale de `main()`

   `main (int argn, char * argv[], char **arge) ;`

   `arge` est un pointeur sur l'environnement

3. `getenv()`

   Fonction de la bibliothèque standard

   `char * getenv (char *var) ;`

   Retourne un pointeur sur la valeur de la variable `var`

Code de retour

- Sous le shell, tester la valeur de retour d'une commande
- Convention :

0	déroulement normal
$\neq 0$	déroulement anormal

- Cette valeur peut être récupérée au niveau du shell :
  - Variable Bourne shell `$?`
  - Variable C-Shell `$status`
- Primitive `exit ()`

```
exit (int n)
```

- Termine proprement le processus
  Code de retour de valeur `n`

Lancement d'une commande shell

- Depuis un programme C
  Lancer une commande shell
- Fonction de la bibliothèque standard

```
system (char *commande) ;
```

- Lance l'exécution d'un processus shell
  Ce processus interprète la commande
- Le processus appelant est interrompu jusqu'à la fin de l'exécution de la commande
- Retourne -1 en cas d'erreur

Les entrées-sorties

Structuration des fichiers Unix

- Fichier Unix ≡ séquence d'octets

(• Fichiers spéciaux)

- Pas de structuration des fichiers au niveau du noyau
- Structuration possible des fichiers suivant les applications
- Exemple : les fichiers textes Unix
  - Leurs lignes se terminent par "*line_feed*" (`LF`)
  - Le noyau Unix ne connaît pas cette convention
  - Il pourrait manipuler des fins de lignes "*carriage_return*" et "*line_feed*" comme sous MS-DOS

Disques

- Le ou les disques physiques
- Partagés en disques logiques (systèmes de fichiers)
- Montage des disques logiques
  - Disque monté → accessible
  - Disques démonté → inaccessible
- Disques logiques
  - Composés de blocs (i.e. 512 ou 1024 octets)
  - Premier bloc (adresse 0) : *bootstrapping*
  - Second bloc (adresse 1) : *super bloc*
    Taille du disque
    Espaces disponibles
    ..., liste d'index
  - 3^e^région : la *i-liste*, liste d'index
    Contient les *i-nœuds* (*i-nodes*)
  - 4^e^région : les fichiers proprement dits
    Contient les données
- La racine d'un disque logique possède donc toujours le i-node n° 2 :

```
% df /usr
Filesystem   512-blocks        Used       Avail Capacity  Mounted on
/dev/rz3g        792124      664414       48496    93%    /usr
% ls -lid /usr
    2 drwxr-xr-x  21 root     system       512 Nov  8  1995 /usr/
```

Les i-nodes

- Code les caractéristiques d'un fichier
- 64 octets :
  - Numéro du propriétaire
  - Numéro du groupe propriétaire
  - Bits de protection (3×3 + set-uid + set-gid + sticky-bit)
  - Numéros des blocs du contenu du fichier
  - Taille du fichiers (en octets)
  - Dates de création, dernière modification, et dernière accès
  - Type du fichier (ordinaire, répertoire, fichier spécial...)

Accès aux données

Numéros des blocs du contenu du fichier

- 13 entrées (i.e. 13 adresses de blocs)
- Les 10 premières entrées : adresses des 10 premiers blocs du fichier

  5120 octets au maximum
- La 11^{e}entrée : adresse d'un bloc

  Ce bloc contient les adresses des 128 blocs suivants

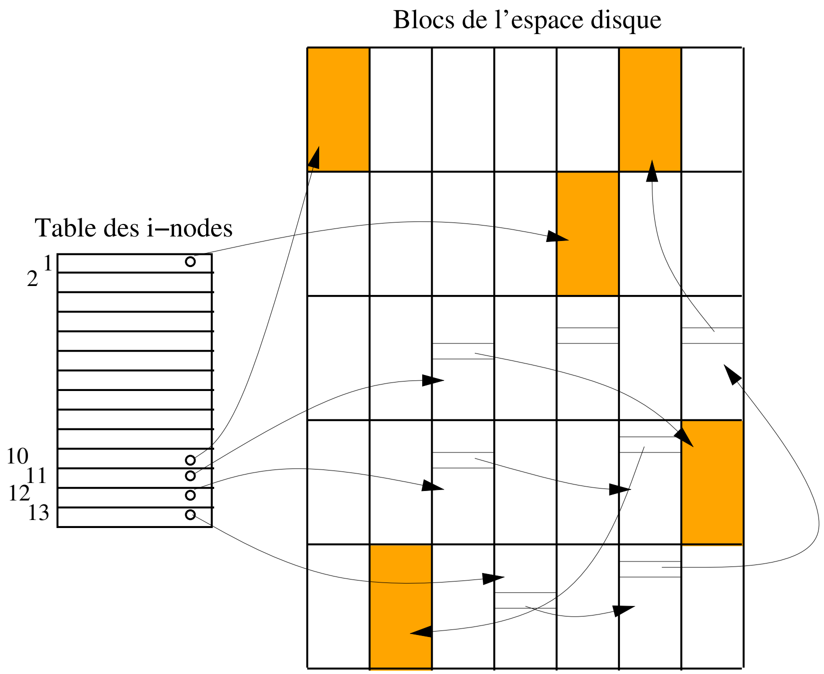
  70656 octets au maximum ((128×512) + 5120)
- La 12^{e}entrée : idem la 11^{e}, avec une indirection supplémentaire

  8.459.264 octets maximum (($128 \times 128 \times 512$) + 70656)
- La 13^{e}entrée : triple indirection

  1.082.201.087 octets au maximum $\approx$ 1 Go

Privilégie les accès aux fichiers de petite taille

Allocation des blocs d'un fichier sur un volume



Structure d'un répertoire

Un répertoire

- Est un fichier
- Son contenu est un tableau
- Ce tableau contient des couples
  - Nom de l'entrée dans le répertoire (`char *`)
  - Numéro d'i-node correspondant à cette entrée
- Deux premières entrées :
  - Le répertoire lui-même (`.`)
  - Son père (`..`)

Appels système et bibliothèque standard

Appels système

- Travail en zone système (cache des fichiers)
- Assure la cohérence des manipulations

La bibliothèque standard

- Travail dans un buffer $\in$ espace de données utilisateur
- Minimise le nombre d'appel au moniteur
- Donc le nombre d'interruption du processus
- Se charge des transferts entre le buffer et le noyau
- À utiliser dans le cas général

Les appels système

- Deux types de référence à un fichier
  1. Noms des fichiers (`char *`)
  2. Descripteur de fichiers (`int`)
- Ouverture : lien entre un nom du système de fichier et un nom local dans le processus (un descripteur)
- Lecture et écriture dans un fichier
- Positionnement dans un fichier
- Définition de synonymie de descripteurs
- Connaissance/modification des caractéristiques d'un fichier
- Fermeture d'un fichier

Les descripteurs

- Nombre maximal de descripteurs par processus

(• Héritage des descripteurs du processus père)

- Trois descripteurs d'entrée-sortie standard
  - Entrée standard : descripteur 0
  - Sortie standard : descripteur 1
  - Sortie d'erreur standard : descripteur 2
- Ces 3 fichiers sont ouverts à la création du processus
- Ne doivent pas être ouvert par l'utilisateur
- Autres fichiers : `open()` retourne un descripteur

Création d'un fichier

```
int creat (char *nom, int mode) ;
```

Ouverture d'un fichier

```
int open (char *path, int oflag, int mode) ;
```

Fermeture d'un fichier

```
int close (int idfi) ;
```

Contrôle des entrées-sorties

```
int fcntl (int idfi, int cmd, int arg) ;
```

Lecture dans un fichier

```
int read (int idfi, char *buffer, unsigned nbytes) ;
```

Exemple : afficher le contenu d'un fichier

```
int fil ;            /* descripteur du fichier */
char buffer [513] ;  /* buffer de lecture */
int l ;

main (int argc, char *argv[])
{
  /* ouverture du fichier */
  if ((fil = open (argv [1], 0)) < 0) {
    perror ("open") ; exit (fil) ;
  }

  /* lecture et affichage */
  for (;;) {
    if (! (l = read (fil, buffer, 512))) break ;
    buffer[l] = NULL ;
    printf ("%s", buffer) ;
  }

  /* fermeture du fichier */
  if (close (fil) {
    perror ("close") ; exit (2) :
  }

  exit (0) ;
}
```


Écriture dans un fichier

```
int write (int idfi, char *buffer, unsigned nbytes) ;
```

Positionnement dans un fichier

```
int lseek (int idfi, long offset, int whence) ;
```

Établissement d'un lien avec un fichier

```
int link (char *nom1, char *nom2) ;
```

Destruction d'un lien ou d'un fichier

```
int unlink (char *nom) ;
```

Droit d'accès pour les créations

```
int umask (int compmode) ;
```

Changement des droits d'accès

```
int chmod (char *nom, int mode) ;
```

Contrôle des droits d'accès

```
int access (char *nom, int mode) ;
```

Changement de répertoire

```
int chdir (char *nomdir) ;
```

Duplication d'un descripteur

```
int dup (int idfi) ;
```


Structure d'un i-node

- Différentes informations contenues dans un i-node
- Définition d'une `struct` C contenant ces informations :
  Fichier `/usr/include/sys/stat.h`

```
struct  stat {
  dev_t st_dev;         /* ID of device containing a directory*/
                        /*   entry for this file.  File serial*/
                        /*   no + device ID uniquely identify */
                        /*   the file within the system */
  ino_t st_ino;         /* File serial number */
  mode_t st_mode;       /* File mode */
  nlink_t st_nlink;     /* Number of links */
  uid_t st_uid;         /* User ID of the file's owner */
  gid_t st_gid;         /* Group ID of the file's group */
  off_t st_size;        /* File size in bytes */
  time_t st_atime;      /* Time of last access */
  time_t st_mtime;      /* Time of last data modification */
  time_t st_ctime;      /* Time of last file status change */
  uint_t st_blksize;    /* Size of block in file */
  int st_blocks;        /* blocks allocated for file */
};
```

- Les dates : secondes depuis le 1^er^janvier 1970
- Les types employés dans `struct stat` dans `/usr/include/sys/types.h`
- Utilisation dans un source C :

```
#include <sys/types.h>
#include <sys/stat.h>
```

- Noms symboliques et constantes dans `/usr/include/sys/stat.h`

Information sur un fichier ouvert

```
int fstat (int idfi, struct stat *buffer) ;
```


La bibliothèque

- Notion de *stream*
- Fonctions
  - `fopen ()`, `fclose ()`, `fread ()`, `fwrite ()`, `fseek ()`...
  - `getc ()`, `fgets ()`, `putc ()`, `fputs ()`...
  - `fprintf ()`, `fscanf ()`...
- Voir le cours de C

Les processus

1. Gestion des processus
2. Communication entre processus

- Voir le chapitre “Notions de processus” et le cours de Maîtrise

Quelques bibliothèques

- Allocation dynamique
  `man malloc`
- Manipulation de chaînes de caractères
  Voir cours de C !
  `man string`
- Générateur de nombres aléatoires
  `man rand`
- Manipulation du fichier `/etc/passwd`
  `man getpwent`
- Manipulation du temps
  `man gettimeofday`

- Fonctions mathématiques

  Voir cours de C ou le manuel en ligne !

fonction	nom	type	# args	type args
valeur absolue	abs	int	1	int
	fabs	double	1	double
exponentielle	exp	double	1	double
logarithme	log	double	1	double
	log10	double	1	double
plus petit majorant	ceil	double	1	double
plus grand minorant	floor	double	1	double
mantisse et exposant	frexp	double	2	double
				int *
distance euclidienne	hypot	double	2	double
puissance	pow	double	2	double
racine carrée	sqrt	double	1	double
trigonométrie	sin	double	1	double
	cos	double	1	double
	acos	double	1	double
	acos	double	1	double
	atan	double	1	double
	atan2	double	2	double
hyperbolique	sinh	double	1	double
	cosh	double	1	double
	tanh	double	1	double

OUTILS UNIX

Contents

Unix conçus pour le développement

- Unix a été conçu
  - par des développeurs
  - pour des développeurs
- De nombreux outils de développement disponibles

Édition

- ed Éditeur de base
- vi Éditeur visuel
- emacs Éditeur GNU
- spell Correcteur orthographique
- eqn Équations
- troff Production de doc (par exemple man)
- TEX, LATEX idem
- ...

Programmation

- cpp Pre-processeur C
- cc, gcc Compilateurs C
- f77, f90 Compilateurs Fortran
- ld Éditeur de liens
- ar, ranlib Archives de bibliothèques
- lint Verificateur de programmes C
- cb, indent C beautifier, indenteur
- dbx Debugger au niveau des sources
- lex Générateur d'analyseurs lexicaux
- yacc Yet another compiler compiler
- make, sccs

`make`

- Maintenance, mise à jour et regénération de fichiers dépendants
- Sources → exécutables
- Recompilation quand nécessaire (dates)
- Fichier de règles de dérivation
  `Makefile` ou `makefile`
- Appel

  `make targets`

Format d'une règle

- Quoi, pourquoi, comment

```
target:  dependencies
         commands...
```

Quoi `target`

Objectif

Généralement un fichier

Pourquoi `dependencies`

Liste des fichiers/cibles dont dépend `target`

Comment `commands`

Précédées d'une tabulation `\t`

Exemple de Makefile

```
#
# makefile pour un programme C
#

executable: f1.o f2.o
        cc -o executable f1.o f2.o

f1.o: f1.c fichier.h
        cc -c f1.c

f2.o: f2.c fichier.h
        cc -c f2.c

clean:
        rm -f *~ *.o executable

# et voila !
```


Algorithme de make

- Pour chaque cible
  - Vérifier les dépendances
    $\rightarrow$ Récursion
    $\rightarrow$ Date des fichiers de base
  - Si modification
    alors $\rightarrow$ Lancer les commandes
    fsi
    sinon $\rightarrow$ Fichier à jour

Commodités

- Commentaires

  `# this is a comment`
- Inclusion de fichiers `Makefile` : `include` *`file`*
- Cible par défaut : `all`

  `all: executable`
- Macros dans le style des variables `sh`
- Cibles spéciales
- Règles implicites

Macros dans un `makefile`

Facilitent la portabilité

- Définition

  `MACRO=valeur`
- Exemples :

  `CFILES=*.c`

  `OBJECTS=fichier1.o fichier2.o`
- Référence :

  `$(MACRO)`
- Macros prédéfinies
- Macros spéciales

Macros prédéfinies

- Exemples pour C
  `$(CC)` compilateur C
  `$(CFLAGS)` options pour le compilateur C
  `$(LINT)` pour lint
- Autres langages
  `$(FC)` Compilateur Fortran
  `$(PC)` Compilateur Pascal
  `$(AS)` Assembleur
- Etc.
  `$(RM)` = `rm -f`

Macros spéciales

- Récursion dans les règles
  `$(MAKE) targets`
- `$(MAKEFLAGS)`
- `$(SHELL)` : interpréteur pour les commandes
  défaut : `/bin/sh`
- `$(HOST_ARCH)` architecture de la machine
  `$(TARGET_ARCH)`, `$(HOST_MACH)`, `$(TARGET_MACH)`

Cibles spéciales

- `.INIT` : cible préliminaire
- `.DONE` : cible ajoutée après les cibles
- `.IGNORE` : ignore les erreurs
- `.SILENT` : exécution silencieuse
- `.PRECIOUS` : fichiers à ne pas effacer en cas d'échec
- `.SUFFIXES` : liste des suffixes pour les règles implicites
- `.PHONY` : entrées qui n'est pas un fichier
- Exemple :

```
.PHONY : clean
clean:
        rm -f *~ *.o executable
```

⇒ make n'est pas perturbé

  - par un eventuel fichier `clean`
  - si `rm` retourne une erreur

Règles implicites

- Modèle de règle
- Application si suffixes correspondent
  `.c` → `.o`
- Macros spéciales
  `$<` fichier de dépendance
  `$@` fichier cible
  `$*` nom de base de ces fichiers

Exemple makefile

```
#
# un makefile plus joli, plus propre, ...

all: executable

OBJECTS = f1.o f2.o

.SUFFIXES: .c .o

.c.o: fichier.h
	$(CC) $(CFLAGS) $<

executable: $(OBJECTS)
	$(CC) $(CFLAGS) -o executable \
	                $(OBJECTS)
.PHONY: clean
clean:
	$(RM) $(OBJECTS) executable *~

# et voila !
```


Remarques

- Options de `make`

  `-s` : silent

  `-d` : affiche les dépendances quand recompile

  ...
- Commandes shell :

  accès aux variables shell par `$$`

  continuation `\` nécessaire
- Génération automatique de `Makefile`

Exemple makefile et sh

```
# un makefile avec du shell

all: executable

SOURCES = *.c
.SILENT:

lignes:
        echo "nombre de lignes"
        for file in $(SOURCES) ; \
        do \
         echo "$$file = `wc -l $$file `";\
        done

informations:
        $(MAKE) lignes
        ...

# et voila !
```


SCCS : source code control system

- Gestionnaire de versions de fichiers
- Regroupe divers utilitaires
- Fonctions :
  - Historique des versions
    Enregistre les différentes versions
    Quels changements, pourquoi, par qui, quand
    Informations gardées dans un “s-file”
  - Vérouillage des fichiers édités
    ⇒ développement par plusieurs persones
- Outils standard Unix : SCCS
  Il existe aussi
  - RCS : produit GNU, fonctionnalite identique à SCCS
  - CVS : Concurrent Versions System

Opérations de base

Get – Récupérer un fichier pour compilation (pas d'édition)

Récupère une version d'un fichier depuis un s-file

Par défaut, la dernière version

Pas de changement du fichier possible (droit `r--`)

Edit – Récupérer une version pour édition

But : éditer le fichier puis enregistrer les modifications dans le s-file

Récupère une version d'un fichier depuis un s-file

Par défaut, la dernière version

Une seule personne peut éditer un fichier à un instant donné (garanti par SCCS)

Put – Enregitrer les modifications dans le s-file

Un nouveau numéro de version est généré

Un commentaire doit être ajouté pour expliquer la modification

Terminologie

S-file C'est un fichier unique qui conserve :

1. Toutes les versions d'un des fichiers utilisateur
   (seules les différences sont stockées ⇒ gain de place)
2. Les informations complémentaires
   Commentaires, date...

Il est stocké dans un sous-répertoire (par défaut `SCCS`)

Son nom est `s.filename`

Delta C'est l'ensemble des changements d'une version à l'autre

Le s-file contient l'ensemble des deltas

SID SCCS Id

Nombre qui représente un delta

Généralement 2 niveaux : `release.level`

Mots clés Identification d'un fichier

Lors d'un *get* (⇒ compilation/installation)

Mots clés sont expansés par SCCS

Exemple : numéro de versions...

Mots clés de la forme `%x%`, par exemple :

`%I%` SID du dernier delta appliqué
`%Z%` Marque `@(#)`, utilisée par la commande `what`
`%W%` Nom du module + SID + date
`%G%` Date du dernier delta appliqué

Création de fichiers SCCS

Csh shell script :

```
% mkdir SCCS save
% foreach file (*.[ch])
    sccs admin -i$file $file
    mv $file save/$file
  end
```

- Crée les s-file associés aux fichiers précisés
- Détruit les fichiers (sauvegarde dans `save`, au cas où...)
- Si l'on veut des mots clés dans les fichiers, mieux vaut le faire avant de créer les s-file

Récupérer les fichiers pour la compilation

- Pour obtenir une copie de la dernière version :

  ```
  sccs get prog.c
  ```

  SCCS répond

  ```
  1.1
  87 lines
  ```

- La version 1.1 a été récupérée, elle contient 87 lignes
- Le fichier `prog.c` a été créé dans le répertoire courant
- Droits = lecture seulement

Modifier les fichiers

- Demander une copie pour l'édition :

```
sccs edit prog.c
```

  ou

```
sccs get -e prog.c
```

- Réponse idem à `get`, en plus :

```
New delta 1.2
```

- Édition du fichier

```
emacs prog.c
```


Enregistrer les changements dans le s-file

- Une fois les changements réalisés dans le fichier
- Enregistrer les changements dans SCCS

```
sccs delta prog.c
```

- Cette commande demande interactivement le commentaire à insérer avec le delta dans le s-file
- Et affiche

```
1.2
5 inserted
3 deleted
84 unchanged
```

- Le fichier `prog.c` peut maintenant être détruit
  Il sera reconstruit par un `sccs get` si nécessaire

Quand réaliser un delta ?

- Pas avant qu'une compilation et un test aient été effectués
- Exemple d'utilisation :
  - `Get` tous les sources
  - `Edit` les sources nécessaires
  - Réalisation de changements, tests, compilation
    Jusqu'à obtention du résultat voulu
  - `Delta` de tout ce qui a été édité
  - Re-`get` tous ce qui a été édité
  - Recompilation totale

Information

- Trouver tous ce qui est en cours d'édition :

  ```
  sccc info
  ```

  Affiche nom du fichier, version, nom utilisateur, date...
- La commande

  ```
  sccs check
  ```

  retourne une valeur non-nulle si une édition a lieu
  ⇒ utilisation dans un script
  Par exemple abandonner une installation si tout n'a pas été correctement passé par delta
- Si tout ce qui a été édité doit être passé à delta :

  ```
  sccs delta `sccs tell`
  ```

- Option `-u [user-name]`

  ```
  sccs -u marquet  info
  sccs -u check
  sccs -u $USER tell
  ```

Les mots-clés SCCS

- Les mots-clés insérés dans les fichiers sont expansés par `get`
- Exemple

```
static char SccsId[] = "%W%, %G%" ;
```

sera remplacé par

```
static char SccsId[] = "@(#)prog.c 1.2, 01/20/94" ;
```

- La commande `what`

```
sccs what prog.c ../usr/bin/prog
```

retrouve le `@(#)` et imprime

```
prog.c:
        prog.c  1.2       01/20/94
../usr/bin/prog
        prog.c  1.1       01/12/94
```

L'exécutable est donc antérieure à la dernière version

- Pour ce faire, le `@(#)` doit apparaître dans l'exécutable
  Donc dans une chaîne de caractères déclarée statique
- Pour les fichiers d'entête (`.h`) → dans un commentaire

Projets multifichiers

- Mettre à jour simultanément tous les fichiers d'une application
- Éditer tous les fichiers

```
sccs edit SCCS
```

- Une fois les modifications effectuées
- Faire un `delta` sur tous ces fichiers

```
sccs delta SCCS
```

Un commentaire unique pour tous les fichiers

Une nouvelle release

- Option `-r`

  `sccs edit -r2 prog.c`

  La nouvelle version sera numérotée 2.1

- Pour tous les fichiers

  `sccs edit -r2 SCCS`

Les anciennes versions

- Option `-r`

  `sccs get -r1.2 prog.c`

- Option `-c`

  `sccs get -c940119120000 prog.c`

  La version courante au 94/01/19 à 12:00:00

SCCS et `make`

- Entrées du `Makefile`
  - `all` Tout reconstruire
  - `sources` Contruire tous les sources à partir des fichiers SCCS
  - `clean` Détruire tout ce qui peut être reconstruit

    `sccs clean`

    Détruit tous les fichiers pour lesquels un s-file existe
  - etc.

- Maintenir un simple programme

```
LDFLAGS= -s -i

prog: prog.o
        $(CC) $(LDFLAGS) -o prog prog.o
prog.o: prog.c prog.h

exmpl: exmpl.o
        $(CC) $(LDFLAGS) -o exmpl exmpl.o
exmpl.o: exmpl.c

.DEFAULT:
        sccs get $<
```

Si un fichier n'existe pas, la dernière règle est appliquée

On tente de le construire à partir d'un fichier SCCS

- Autre possibilité

```
SRCS= prog.c prog.h exmpl.c
LDFLAGS= -s -i

prog: prog.o
        $(CC) $(LDFLAGS) -o prog prog.o
prog.o: prog.h

exmpl: exmpl.o
        $(CC) $(LDFLAGS) -o exmpl exmpl.o

sources: $(SRCS)
$(SRCS):
        sccs get $@
```

Avantages :

1. Pas de dépendances explicites entre le `.o` et le `.c`
2. Il existe une entrée `sources`
3. Le `Makefile` ne tentera pas de faire des `get` sur des fichiers inexistants

- Le suffixe ~

  Les fichiers SCCS n'ont pas de suffixe particulier

  Mais un préfixe (SCCS/s.)

  Convention dans make : le suffixe ~ désigne le s-file

  Règles par défaut

```
.c~.o:
        $(GET) $(GFLAGS) $<
        $(CC) $(CFLAGS) -c $*.c
        -$(RM) -f $*.c
```


Compression de fichiers par compress

```
compress filename
uncompress compressedfile
zcat compressedfile
```

- Outils Unix standard
- Extension .Z des fichiers compressés
-

file	→ compress →	file.Z
file.Z	→ uncompress →	file
file.Z	→ zcat →	stdout

- compress et uncompress détruisent le fichier source
- zcat produit sont résultat sur stdout

```
% ls -l bigfile*
-rw-r--r--   1 marquet  153600 Sep 20 09:31 bigfile
% compress bigfile
% ls -l bigfile*
-rw-r--r--   1 marquet   56218 Sep 20 09:33 bigfile.Z
% zcat bigfile.Z > bigfile2
% ls -l bigfile*
-rw-r--r--   1 marquet   56218 Sep 20 09:33 bigfile.Z
-rw-r--r--   1 marquet  153600 Sep 20 09.33 bigfile2
```

Compression de fichiers par gzip

```
gzip [-best] filename
gunzip compressedfile
zcat compressedfile
```

- Alternative GNU à la famille compress
- Extension .gz des fichiers compressés
- Lit aussi les fichiers au format .Z

```
% gzip -best bigfile2
-rw-r--r--   1 marquet   56218 Sep 20 09:33 bigfile.Z
-rw-r--r--   1 marquet   38550 Sep 20 09:44 bigfile2.gz
```

- Il existe aussi bzip2, Extension .bz2

Encryption de fichiers binaires uuencode

```
uuencode file name
uudecode filename
```

- Utilisés pour transmettre des fichiers binaires via des medium qui ne supportent que les simples caractères ASCII

  Par exemple : transmettre un fichier compressé par mail
- Extension conventionnelle .uu ou .uue
- file, name → uuencode → stdout

  file.uu → uudecode → name
- Le nom name est mémorisé dans le fichier .uu

```
% uuencode bigfile.Z bigfile.Z > bigfile.Z.uu
% head -3 bigfile.Z.uu
begin 644 bigfile.Z
M'YV03<*L*6,F#9LR`!(J7,BPH<.'$"-*G$@1`(R+-FCO`&$QA@R-,#A>Q&A#
MY,B+,6+0B'$#!`P;,F3`T"CCAHV0,&+,^%C28L6?0(,*'0J@SAPZ8>18A-$F
% tail -3 bigfile.Z.uu
MWP.=X/?H)Z@SWPC5'.<+YT_[JDZ]YG&?\+=^SX_]@%'12T)U/AAS_@X%C;7/
M-W_GY%OF2/*[]V%Z^!.<+
end
```

Manipulation de fichiers archives tar

```
tar c[zv]f tarfile filenames...
tar t[zv]f tarfile
tar x[zv]f tarfile
tar ...
```

- Création, listage ou extraction de plusieurs fichiers sur un fichier unique

  Fichier d'archive

  (– Bandes magnétiques)
  - Fichier quelconque
  - Option `f tarfile`
  - Option `f -` ⇒ `stdin/stdout`
- Option `v` : mode verbose
- Option `z` : (GNU `tar`) compression `gzip`
- Options `ctx` : mode création/listage/extration


```
% ls -R
Makefile   README     doc/       include/   src/
./doc:
conn.man   conn.txt
./include:
defs.h   exit.h
./src:
conn.c   main.c   resd.c
% tar cf All.tar *
% gzip -best All.tar
% ls
All.tar.gz  README      include/
Makefile    doc/        src/
% zcat All.tar.gz | tar tf -
Makefile
README
doc
doc/conn.txt
doc/conn.man
include
include/defs.h
include/exit.h
src
src/conn.c
src/main.c
src/resd.c
```

```
% cd ...
% tar xzvf All.tar.gz
Makefile
README
doc
doc/conn.txt
doc/conn.man
include
include/defs.h
include/exit.h
src
src/conn.c
src/main.c
src/resd.c
```


ASPECTS RÉSEAU

Contents

Aspects réseau d'Unix

- Machine Unix connectée à un réseau
- Accès à distance (CPU, fichiers..)

  Commandes réseau
- Partage de ressources (fichiers, systèmes de fichiers, imprimantes...)

  Architecture distribuée du système Unix

Réseau TCP/IP

- Réseau : ensemble d'*hôtes* capables de communiquer entre eux
- *Hôtes* : ordinateurs, terminaux X, imprimantes...
- Réseau TCP/IP
  - Projet de recherche 1969 de la DARPA
    Réseau ARPANET opérationnel en 1975
  - Protocoles TCP/IP : standard en 1983
  - ARPANET $\longrightarrow$ *Internet*
  - TCP/IP aussi utilisé par les réseaux locaux Unix
  - Exemple :
    - ▷ Formations en Informatique de Lille
      Réseau local de machines Unix
    - ▷ Université de Lille 1
      Réseau fédérant les réseaux locaux des différents laboratoires
      Relié à l'Internet par une ligne (unique) à haut débit
  - Non-exclusivement Unix
  - Nommage des machines
    `msc12.fil.univ-lille1.fr`
  - Supports physiques variés
    Le plus courant/moins coûteux : *Ethernet*

Nommage des machines

Numéro Ethernet – Adresse IP

- Une machine possède une interface réseau
- Numérotation de ces interfaces réseau : un numéro *Ethernet*
  - Adresse sur 6 octets
    Exemple : `08:00:20:05:39:60`
  - Adresse unique associée au matériel

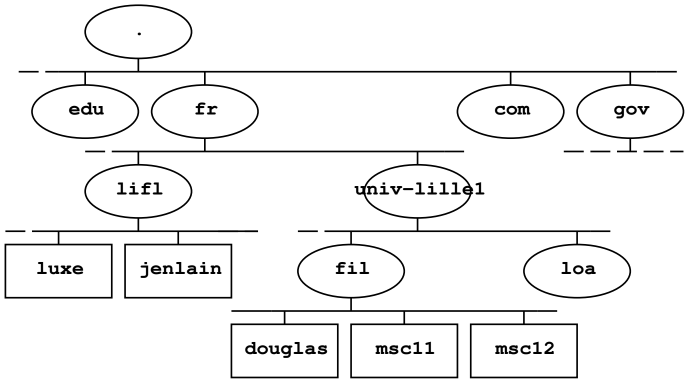
Préfixe	Constructeur
`10:00:5A`	IBM
`08:00:09`	HP
`AA:00`	Digital
...	...

- Machine connectée à l'Internet : numéro IP (*Internet Protocol*)
  - Adresse 32 bits
  - Notation pointéee
    `134.206.12.26`

  Numéro IP attribué par une autorité centrale (le NIC)
- Correspondance entre le numéro IP et le numéro Ethernet : ARP (*Address Resolution Protocol*)
  - Diffusion : `Qui est 134.206.12.26`
  - Réponse : `Moi :  08:00:20:05:39:60`
  - Cache des réponses récentes
- Accès à la machine $\Rightarrow$ son numéro IP

Adresse IP – Nom de machine

- Accès à la machine ⇒ son numéro IP
  Mémorisation des numéros difficile ⇒ nommage des machines
- Correspondance entre le nom de la machine et numéro IP : DNS (*Domain Name System*)
- Organise les noms de machines en hiérarchie de domaines



- Gestion distribuée de la table de correspondance entre les noms et les numéros IP
- Délégation hiérarchique de la gestion de ces noms

- Noms complètement qualifiés
  `nom-de-machine . nom-de-sous-domaine .`
  `nom-de-domaine . nom-de-`*`top-level`*`-domaine`
  `msc12.fil.univ-lille1.fr`
- Noms non complètement qualifiés
  `msc12`
- Retrouver un numéro IP à partir d'un nom ?
  1. Fichier `/etc/hosts`
     Résolution statique
     Exemple :
     `134.206.12.26 msc4.fil.univ-lille1.fr msc4`
  2. Interrogation d'un serveur DNS
     Résolution dynamique
     2 % de la charge du réseau

Commandes réseau

- Accès à des machines distantes
  - CPU : connexion à distance
  - Fichiers : copie entre machines
- Deux types de commandes :
  - Commandes spécifiques aux machines Unix
    Commandes Bekerley `rlogin`, `rsh`, `rcp`...
  - Communications avec des systèmes hétérogènes
    Commandes ARPA `ftp`, `telnet`

Connexion à distance sur une machine Unix

`rlogin [-l username] hostname`

- Permet le login
  - sur une machine Unix *distante* (par exemple `douglas`)
  - depuis une machine *d'origine* (`msc12`)

```
msc12% rlogin douglas
Password:
douglas%
```

- Par défaut contrôle du mot de passe
- Non contrôle du mot de passe
  - Brèche de sécurité
  + Mise en échec des renifleurs
- Non contrôle du mot de passe
  - Machine équivalente
  - Utilisateur équivalent

- Non contrôle du mot de passe
  - Machine équivalente
    - ▷ La machine origine est listée dans le fichier `/etc/hosts.equiv` de la machine distante
    - ▷ Même nom d'utilisateur (pas d'option `-l`)
  - Utilisateur équivalent
    - ▷ Fichier `.rhosts` dans `$HOME` de l'utilisateur sur la machine distante contient une ligne

      `machine-origine [user-origine]`

      Joker +
    - ▷ Exemple :

```
msc12% rlogin douglas
douglas% cat .rhosts
msc12 marquet
msc12.fil.univ-lille1.fr marquet
```

    - ▷ Ce fichier doit appartenir à l'utilisateur distant

- Contrôle de la session `rlogin` (C-shell)
  - `~^Z` suspend la session `rlogin`

```
msc12% rlogin douglas
Password:
douglas% ...
douglas% ~^Z
[1] + Suspended rlogin douglas
msc12% ...
msc12% fg
rlogin douglas
douglas%
```

  - Terminaison de la session `rlogin`
    - ▷ À la terminaison du shell : `logout`, `^D`
    - ▷ Par rupture de la connexion : `~.`
  - Caractère d'échappement `~` peut être choisi

    Option `-e`*`caractère`* de `rlogin`

Exécution à distance sur une machine Unix

`rsh [-l username] hostname [commande [argument...]]`

- Exécution de la commande (avec ses arguments) sur la machine Unix distante
- Ne fonctionne que si mécanisme d'équivalence (`/etc/hosts.equiv` ou `$HOME/.rhosts`)
- Exemples

```
msc12% rsh douglas who
duchmol tty1 Sep 12 09:05
lefebvre ttyp1 Sep 12 07:45
lefebvre ttyp0 Sep 12 07:47
msc12% tar cf - . | rsh douglas tar xf -
```

- Répertoire de travail a distance = répertoire de login
- Attention aux arguments de la commande

  Différence entre les deux commandes

```
msc12% rsh douglas cat file-d ">>" file-distant.all
msc12% rsh douglas cat file-d >> file-local.all
```

- Exécute le contenu du fichier `.cshrc` sur la machine distante (C-shell)

  Pas d'exécution du `.login`
- Si pas de commande ⇒ idem `rlogin`

- Interprétation distante de métacaractères

  ⇒ échappement (`\`, `""`, ou `''`)
- Exemple :

```
msc12% ls /tmp/*.c
tfr.c tus.c
msc12% rsh douglas
douglas% ls /tmp/*.c
ar.c ax.c
douglas% ^D
rlogin: connection closed.
msc12% rsh douglas ls /tmp/*.c
ls: /tmp/tfr.c: No such file or directory
ls: /tmp/tus.c: No such file or directory
msc12% rsh douglas "ls /tmp/*.c"
ar.c ax.c
```

Travail à distance sur une machine Unix

- Connexion à distance (`rlogin`, `telnet`)
- Application en mode graphique sur la machine distante
- Interface X11 (R6)
- Principe :
  - Serveur X11 sur la machine locale
  - Applications : clients X11
  - Communication entre les le client et le serveur via Internet
- Sur la machine locale (`msc12`) :
  - Autoriser le serveur X11 à servir les clients de la machine distante :

```
msc12% xhost +douglas
douglas being added to access control list
```

- Sur la machine distante (`douglas`) :
  - Indiquer aux clients de se connecter au serveur de la machine locale (`csh`) :

```
douglas% setenv DISPLAY msc12:0
```

  - Lancer les clients X11 :

```
douglas% xeyes &
[1] 4342
douglas% emacs &
[2] 4344
```


Copie de fichiers entre machines Unix

`rcp [-r] source destination`

- La syntaxe des fichiers `source` ou `destination` est la suivante :

```
filename
[user]@host:filename
host[.users]:filename
```

- Les `filename` distants sont interprétés relativement au répertoire de login de l'utilisateur distant
- Mécanisme d'équivalence nécessaire (`/etc/hosts.equiv` ou `$HOME/.rhosts`)
- Exemple :

```
msc12% rcp douglas:/tmp/ax.c .
```

- Option `-r` : copie récursive
- Échappement des métacaractères devant être interprétés à distance :

```
msc12% rcp "douglas:/tmp/*.c" .
```

Connexion à distance entre machines Unix ou non

`telnet host`

- Connexion de même manière que `rlogin`
- Pas d'utilisation des fichiers `/etc/hosts.equiv` et `$HOME/.rhosts`
- Nom de login et mot de passe introduits lors de la connexion
- Exemple :

```
msc12% telnet douglas
Trying 134.206.12.22...
Connected to douglas.fil.univ-lille1.fr.
Escape character is '^]'.

UNIX(r) System V Release 4.0 (douglas)

login:
```


Transfert de fichiers

- Commande `ftp`
- Altenative : `ncftp`
- Nombre limité de commandes sur le système distant
- Ces commandes sont filtrées par le système distant qui les transforme en commandes adaptées au système d'exploitation de la machine
- Commandes disponibles :
  - Transfert de fichiers (`put`, `get`...)
  - Listage de répertoire (`ls`, `dir`...)
  - Effacement de fichier (`delete`...)
  - Changement de répertoire (`cd`, `pwd`...)
  - Etc...
- Commandes sur la machine locale : `lcd` et préfixe !
- Exemple de session `ftp`

  Aller chercher
  - le fichier `ax.c`
  - dans le répertoire `/tmp`
  - de la machine `douglas.fil.univ-lille1.fr`

  Le compiler localement

  Retourner le résultat sur la machine distante

```
msc12% ftp douglas.fil.univ-lille1.fr
Connected to douglas.fil.univ-lille1.fr.
220 douglas FTP server (UNIX(r) System V Release 4.0) ready.
Name (douglas.fil.univ-lille1.fr:marquet): marquet
331 Password required for marquet.
Password:
230 User marquet logged in.
Remote system type is Unix.
ftp> cd /tmp
250 CWD command successful.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for file list.
ar.c ax.c
226 Transfer complete.
ftp> lcd tmp
Local directory now /home/enseign/marquet/tmp
ftp> get ax.c
200 PORT command successful.
150 Opening ASCII mode data connection for ax.c (4932 bytes).
226 Transfer complete.
4932 bytes received in 0.37 seconds (13 Kbytes/s)
ftp> !cc ax.c -o ax
ftp> bin
200 Type set to I.
ftp> put ax
200 PORT command successful.
150 Opening BINARY mode data connection for ax (87338 bytes).
226 Transfer complete.
87338 bytes sent in 1.9 seconds (45 Kbytes/s)
ftp> quit
221 Goodbye.
```


- Nom de login et mot de passe ⟶ pseudo-login
- Ftp anonyme :
  - Utilisateur distant sans mot de passe (anonymous)
  - Mot de passe : adresse email
  - Droits réduits
  - Répertoire /pub, fichier README...

```
msc12% ftp ftp.univ-lille1.fr
Connected to beta.univ-lille1.fr.
220 beta Microsoft FTP Service (Version 1.0).
Name (ftp.univ-lille1.fr:marquet): anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
Password: marquet@lifl.fr
230-NEW: You are on a brand new server
 If you find some problems, please refer
 them to ftpmaint@univ-lille1.fr
 Doing so you will help us to provide you a better service.
 Thanks.
230 Anonymous user logged in as IUSR_BETA.
Remote system type is Windows_NT.
ftp> dir
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
---------- 1 owner group   2773877 Aug  9 11:13 ls-lR.Z
---------- 1 owner group      6296 Aug 12 10:06 MIRRORS
d--------- 1 owner group       512 Sep  5 13:30 pub
---------- 1 owner group      1660 Apr 19  8:50 README
226 Transfer complete.
```

Architecture distribuée d'Unix

- Services NIS
- Serveur de fichiers NFS
- Autres services
  - Serveur d'impression BSD (`lpd`)
  - Serveur de nommage DNS
  - Serveur de messagerie
  - Serveur de pages `html`
  - Synchronisation d'horloges (`rdate`, `ntpd`)

Service NIS

- *Network Information Services*
  Ex- “Yellow pages”
- Serveur de fichiers d'administration (fichiers de `/etc`)
  - `/etc/passwd` : utilisateurs & mots de passe
  - `/etc/group` : groupes d'utilisateurs
  - `/etc/hosts` : noms et numéros des machines
  - Etc
- Gestion centralisée des fichiers par le serveur NIS
- Notion de domaine NIS : ensemble des machines servies
- Diffusion par le serveur aux autres machines des fichiers
- Quel est le serveur pour une machine donnée ?

```
msc14% ypwhich msc14
petrus.fil.univ-lille1.fr
```

- Lister un fichier géré par NIS

```
msc14% ypcat passwd | head -3
duchmol:641qXHzHthhjk:7713:77:Pierre Duchmol:/home/licence/duchmol:/bin/c
marquet:UCRtTDCmnJEJM:7333:73:Philippe Marquet:/home/enseign/marquet:/bin
albert:ruFx8ojbEQgMM:7507:75:Francis Albert:/home/maitrise/albert:/bin/cs
```

- Commandes de mise à jour

  Par exemple `passwd`

  ⇝ `yppasswd` ou `nispasswd` ou `passwd` !

```
msc14% yppasswd
yppasswd:  Changing password for marquet
Enter login(NIS) password:
New password:
Re-enter new password:
NIS(YP) passwd/attributes changed on petrus.fil.univ-lille.fr
```


Serveur de fichiers NFS

- *Network File System*
- Système de fichiers distribués par le réseau Ethernet
- À l'origine : produit Sun
- Transparent à l'utilisateur

  Arborescence Unix inchangée
- Montage d'un système de fichiers d'un serveur NFS sur la machine Unix locale
- Droits : UID et GID

  Et non nom de login

  ⇒ UID/GID identique sur le serveur et la machine locale

  ⇒ Administration centralisée de `/etc/passwd` et `/etc/group` : NIS

```
msc14% mount
/ on /dev/dsk/c0t3d0s0 read/write/setuid on Tue Sep 17 14:34:57 1996
/usr on /dev/dsk/c0t3d0s6 read/write/setuid on Tue Sep 17 14:34:57 1996
/var on /dev/dsk/c0t3d0s3 read/write/setuid on Tue Sep 17 14:34:57 1996
/local on /dev/dsk/c0t3d0s5 setuid/read/write on Tue Sep 17 14:35:00 1996
/tmp on swap read/write on Tue Sep 17 14:35:00 1996
/usr1 on petrus:/usr1 soft/bg/remote on Tue Sep 17 14:35:24 1996
/var/mail on douglas:/var/mail soft/bg/actimeo=0/remote on Tue Sep 17 14:
/home/enseign on petrus:/export/home6/enseign read/write/remote on Tue Se
/opt/gnu on douglas:/opt/gnu read/write/remote on Tue Sep 17 17:45:58 199
```

NOTION DE PROCESSUS

Contents

Processus

- Processus ≡ programme s'exécutant (entité dynamique)
  On parle aussi de tâche
- Unix est multi-tâches
- Un processus utilise des ressources
  Clavier, écran, mémoire, système de fichiers, librairies dynamiques...
- Droits attachés à un processus
  Utilisateur, groupe
- Niveau de priorité
  Modifiable par `nice`
- Arborescence de processus
  Création par `fork()` ou `exec()`
- Environnement associé à un processus : variables (`TERM`...)

Liste des processus

`ps [-aux...]`

- Par défaut, les processus de l'utilisateur
- Options
  - `-a` processus des autres
  - `-u` *`user`* processus de l'utilisateur *`user`*
    Format long par défaut
  - `-e|-A` tous les processus
  - `-f` format long
- Attention à la syntaxe
  `ps aux` (syntaxe BSD) n'est pas équivalent à `ps -aux` !

```
% ps
  PID TTY      S           TIME COMMAND
  563 ttyp1    I  +     0:13.79 -tcsh (tcsh)
 4343 ttyp1    I        0:22.10 xdvi unix-2
 1659 ttyp2    S        0:12.23 -tcsh (tcsh)
 4198 ttyp2    S        4:27.81 emacs
 4337 ttyp4    I  +     0:00.42 rlogin templier
```

Format long

```
% ps -f
USER       PID  PPID %CPU STARTED  TTY             TIME COMMAND
marquet    563   553  0.0   Sep 10 ttyp1        0:13.79 -tcsh (tcsh)
marquet   4343   563  0.0 14:11:21 ttyp1        0:22.10 xdvi unix-2
marquet   1659  1991  0.0   Sep 12 ttyp2        0:12.27 -tcsh (tcsh)
marquet   4198  1659  0.0 13:54:02 ttyp2        4:28.58 emacs
marquet   4337  4332  0.0 13:54:16 ttyp4        0:00.42 rlogin templier
```

- Utilisateur
- PID : numéro unique de processus
- PPID : numéro du processus père
- Statistiques : mémoire, temps CPU, taille...
- État : exécution, swappé, priorité...

Lancement de processus

- Lancement interactif, au prompt :

  `% commande [arguments]`

- Interruption `^Z` (C-shell)
  Arrêt `^C`
- Lancement en tâche de fond :

  `% commande [arguments] &`

  Le shell répond par un numéro de processus (PID)
  (C-shell : et un numéro de *job*, `[2]`) :

  ```
  % emacs &
  [2] 4661
  ```

- Désignation d'un processus
  - Par son PID : `4661`
  - Par son numéro de job (C-shell) : `%2`
- Lancement à une date donnée : `at`, `cron`

Manipulation de processus

- C-shell
- Liste des jobs :

  `jobs [-l]`

- Lancement en tâche de fond ou interactif d'un processus

  ```
  bg [processus]
  fg [processus]
  ```

- Exemple :

```
% jobs
[1]  + Running                 emacs
[2]  - Running                 xrlogin templier
% xdvi unix-2
^Z
[1]  - Running                 emacs
[2]    Running                 xrlogin templier
[3]  + Suspended               xdvi unix-2
% bg %3
[3]    xdvi unix-2 &
% jobs
[1]  + Running                 emacs
[2]  - Running                 xrlogin templier
[3]    Running                 xdvi unix-2
```


Envoyer un signal

`kill -l`

- Liste des signaux

```
% kill -l
HUP INT QUIT ILL TRAP IOT EMT FPE KILL BUS SEGV
SYS PIPE ALRM TERM URG STOP TSTP CONT CHLD TTIN
TTOU IO XCPU XFSZ VTALRM PROF WINCH USR1 USR2
```

`kill [-signal] processus`

- Envoie le `signal` au `processus`
- Signaux :

KILL	9	Termine le processus et ses descendants
STOP	17	Suspend le processus (`^Z`)
CONT	19	Relance l'exécution du processus (`bg`, `fg`)

- Exemple :

```
% jobs
[1]  + Running                 emacs
[2]  - Running                 xrlogin templier
[3]    Running                 xdvi unix-2
% kill -9 %3
[3]    Killed                  xdvi unix-2
% jobs
[1]  + Running                 emacs
[2]  - Running                 xrlogin templier
```

Types de processus

- Gestion de la machine
  `init`, `sync`, `fasthalt`...
- Démons (services, en attente)
  `sendmail`, `lpd`, `cron`...
- Utilisateurs
  - Applications : `emacs`...
  - Interpréteurs : `csh`...
  - Environnement graphique : `twm`...

INDEX