Introduction à UNIX

C. Pham

UPPA
LIUPPA
Université de Pau et des Pays de l'Adour

UNIX et un peu d'histoire

En 1965, Bell, MIT et GE se joingnent pour développer un nouveau système d'exploitation multi-tâches, multi-utilisateurs, multi-processeurs et un système de fichiers multi-niveaux (hiérarchique). Le nom de ce système...MULTICS. En 1969, AT&T décide de se retirer mais certaines personnes qui ont travaillé sur MULTICS continuent sur leur lancée...

... et UNIX ($\neq$ MULTICS) est né aux Laboratoire Bell en 1969. Ken Thompson l'écrit d'abord en langage machine. John Kernighan l'écrit en assembleur. Avec l'apparition du C, UNIX est réécrit en un langage de plus haut-niveau (1973) et reste très intimement lié à ce langage.

En 1974, AT&T autorise les universités à utiliser UNIX à des fins éducationels. Philosophie générale: "small is beautiful", "keep it simple and stupid". La version 7 est commercialisée en 1978.

Combien d'UNIX?

Division en plusieurs souches

- AT&T sort le *System III* en 1978 qui devient *System V* en 1983. En 1989, il y a fusion de *System V* avec SunOS 4.x de SUN.
- L'université de Californie, Berkeley sort UNIX BSD qui évolue à ce jour à BSD 4.4 (1993).
- En 1991, Linux apparait avec un développement indépendent (Linus Torvalds).

Souches issues

- XENIX (Microsoft), HP-UX (HP), UNIX SCO, VENIX... sont issus de AT&T *System V*, .
- NetBSD 1.2 et FreeBSD 2.2 sont tous les deux issus de la souche BSD 4.4.

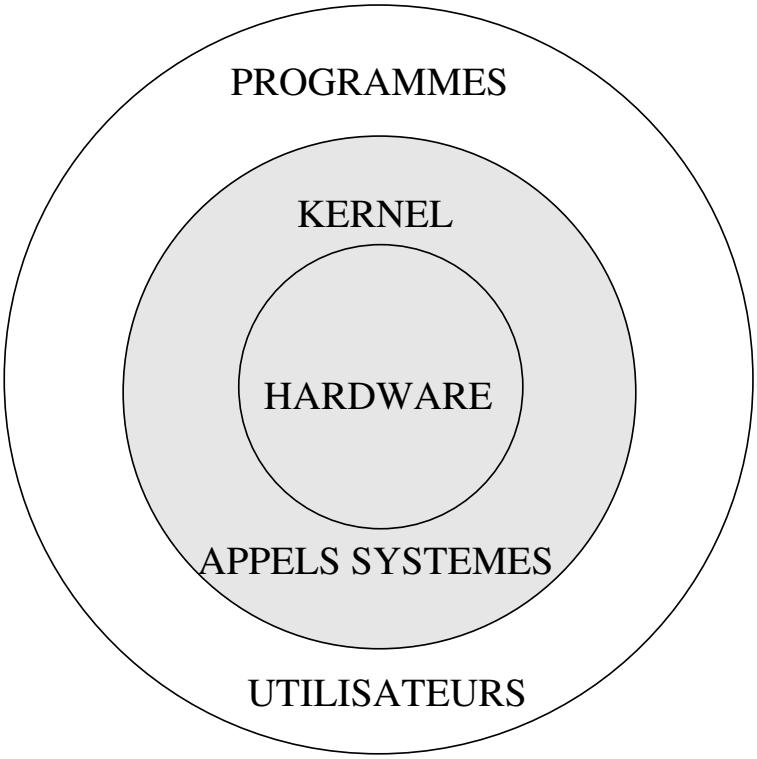
Les principales caractéristiques

UNIX, c'est...

- une hiérarchie (doublée d'une génétique) des processus avec héritage d'un ensemble de caratéristiques d'un processus père à un processus fils,
- la notion que les processus peuvent communiquer entre-eux,
- un multi-tâches préemtif avec notion de priorité,
- des appels systèmes disponibles depuis des langages de haut-niveau,
- des shells permettant d'écrire des scripts et d'enrichir ainsi le noyau de nouvelles commandes,
- un système de fichiers hiérarchisé et généralisé,
- un mécanisme de redirection standard,
- pas de programmes trop généraux, mais plutôt beaucoup de programmes bien spécialisés.

Architecture d'un système UNIX

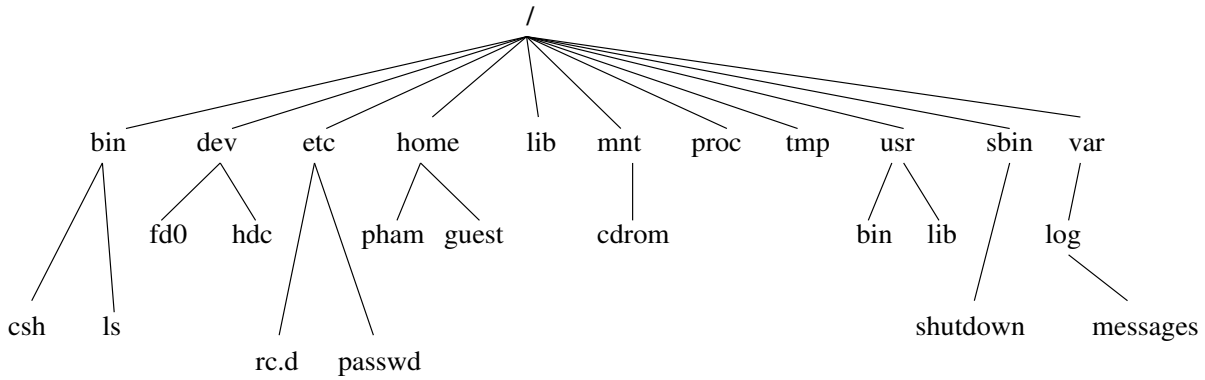
UNIX instaure un cloisonnement strict entre le noyau et la partie réservée aux utilisateurs.



Un certain nombre d'appels systèmes permettent à l'utilisateur d'accéder aux ressources du système. Le but est de préserver l'intégrité du système pour l'utilisateur lui-même, et les autres utilisateurs.

La hiérarchie des répertoires

UNIX définit un système de fichiers hiérarchique avec un certain nombre de répertoires standards.



Il y a cependant des variantes d'une souche à l'autre, mais dans l'ensemble, il n'est pas trop difficile de se repérer lorsque l'on passe sur un autre système.

Le niveau le plus haut est appelé `root`. `/bin` contient généralement les programmes utiles au démarrage et `/etc` les fichiers de configurations. `/dev` contient les fichiers relatifs aux `devices`. `/home` contient les répertoires des utilisateurs. `/lib` contient les librairies du système. `/usr` contient les programmes ajoutés au systèmes, non indispensables.

Se logger

Unix est multi-utilisateur et possède un mécanisme d'identification connu sous le nom de `login` (log in). Pour utiliser un système UNIX sur une machine, il faut avoir un *compte* déclaré accessible sur cette machine. L'invite de login se présente généralement sous la forme suivante:

```
lhpca login: pham
password:
```

A ce stade, l'utilisateur s'identifie en fournissant son mot de passe, qui devrait être seulement connu de lui-même. Si l'identification réussie, l'utilisateur voit apparaître quelque chose comme:

```
lhpca login: pham
password:
Last login: Thu 17 11:22:21 from peg
SunOS Release 4.1.3 (GENERIC)
You have new mail.
peg{1}~>
```

Une mauvaise identification fournie `login incorrect` plutôt que `password incorrect`!

Quelle est mon identité?

Pour UNIX, l'identité d'un utilisateur est celle sous laquelle il se logge. Dans l'exemple précédent, l'utilisateur sera connu sous le nom de `pham`.

Le fichier `/etc/passwd` doit contenir une entrée du type:

```
pham:B9SzS1wt6G:5230:64:Pham CongDuc,UCLA,(310)825-4885,,:/home/pham:/bin/csh
```

L'utilisateur appartient également à un ou plusieurs groupes. Le fichier `/etc/group` peut contenir des lignes du type:

```
users::100:pham
user_::64:pham
```

whoami

```
$ whoami
pham
```

id

```
$ id
uid=5230(pham) gid=64(user_) groups=64(user_),100(users)
```

Quel shell?

Après le login, l'utilisateur accède à un interpréteur de commandes, communément appelé *shell*. Il en existe plusieurs avec des fonctionnalités et des interfaces utilisateurs un peu différentes les unes des autres. Pour savoir quel shell est celui par défaut, taper:

```
% echo $SHELL
/bin/csh
```

chaîne	Shell	Remarque
`/bin/sh`	Bourne Shell	Disponible partout
`/bin/ksh`	Korn Shell	Très puissant
`/bin/csh`	C-Shell	Issu de Berkeley
`/bin/tcsh`	Toronto C-Shell	Dérivé du C-Shell

Le Bourne Shell est le plus ancien. Le C-Shell a une syntaxe proche du langage C. Il y a aussi `/bin/bash` pour *Bourne Again Shell* de GNU et bien d'autres. Un shell restreint (`rsh, rksh`) limite les possibilités de l'utilisateur pour plus de sécurité.

Le shell par defaut est celui qui est indiqué dans le fichier `/etc/passwd`. La liste des shells autorisés se trouve dans le fichier `/etc/shells`.

```
pham:B9SzS1wt6G:5230:64:Pham CongDuc,UCLA,(310)825-4885,,:/home/pham:/bin/csh
```

```
/etc/shells:
            /bin/sh
            /bin/tcsh
            /bin/csh
```

Première commande: `ls`

Une commande c'est un mot-clé avec éventuellement des options. Les commandes sont *case-sensitive*.

```
% commande -[option][option][option]
```

La commande `ls` permet d'afficher le contenu d'un répertoire.

```
% ls
Archive_2   Cours    bin    cdrom    nsmail
tmp
```

Avec l'option `-l`, pour version longue, plus d'information sont affichées.

```
% ls -l
drwxr-xr-x   4 pham   user_    1024 Sep 15 12:40 Archive_2/
drwx------   9 pham   user_    1024 Sep 15 18:00 Cours/
drwxr-xr-x   2 pham   user_    1024 Aug 28 12:08 bin/
lrwxrwxrwx   1 pham   user_      10 Sep 17 11:03 cdrom -> /mnt/cdrom/
drwx------   2 pham   user_    1024 Sep 11 17:02 nsmail/
drwxrwxrwx   2 pham   user_    1024 Sep  7 10:40 tmp/
```

`ls` sur un fichier affiche le nom du fichier si celui existe. Cette commande sur un répertoire affiche le contenu du répertoire, sauf si l'option `-d` est utilisée.

A qui est le fichier?

Un fichier possède un propriétaire et un nom de groupe auquel il appartient.

```
% ls -ld nsmail
drwx------   2 pham   user_    1024 Sep 11 17:02 nsmail/
```

Le propriétaire de `nsmail` est `pham` et ce fichier est aussi dans le groupe `user_`.

On peut changer le propriétaire du fichier avec la commande `chown` et le groupe avec `chgrp`.

```
% chown steve nsmail
% ls -ld nsmail
drwx------   2 steve   user_    1024 Sep 11 17:02 nsmail/

% chgrp etudiant nsmail
% ls -ld nsmail
drwx------   2 steve   etudiant 1024 Sep 11 17:02 nsmail/
```

Si le nouveau groupe possède un mot de passe, l'utilisateur devra le fournir.

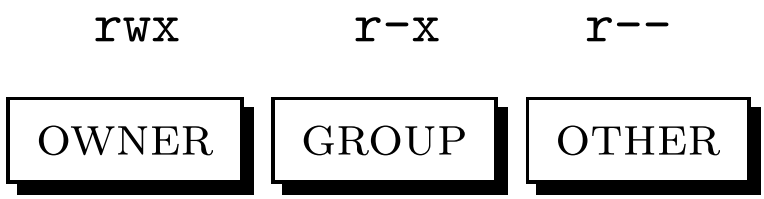
```
users:Hge5Gy:100:pham
user_::64:pham
```

Les attributs d'un fichier

Etant multi-utilisateurs, les fichiers possèdent un certain nombre d'attributs qui définissent les autorisations d'accès. Ces attributs sont regroupés en 3 groupes de 3 attributs.

r	autorisation à lire (read)
w	autorisation à écrire (write)
x	autorisation à l'exécution (execute)

Les 3 groupes sont:



On utilise souvent la réprésentation binaire de ces attributs:

```
rwx   r-x   r--
111   101   100
   \   |   /
       754
```

Modification des attributs

On peut modifier les attributs d'un fichier avec la commande chmod. Les syntaxes possibles sont:

```
% ls -ld nsmail
drwx------   2 pham   user_    1024 Sep 11 17:02 nsmail/

% chmod 777 nsmail
% ls -ld nsmail
drwxrwxrwx   2 pham   user_    1024 Sep 11 17:02 nsmail/

$ chmod g-w,o-wx nsmail
$ ls -ld nsmail
drwxr-xr--   2 pham   user_    1024 Sep 11 17:02 nsmail/

% chmod go=r nsmail
% ls -ld nsmail
drwxr--r--   2 pham   user_    1024 Sep 11 17:02 nsmail/
```

En résumé

+	ajoute la permission
-	retire la permission
=	met la permission à

Navigation à travers les répertoires

pwd cd mkdir

```
% mkdir Paper/tex
% cd Paper/tex
% pwd
/home/pham/Paper/tex
```

Chaque répertoire contient 2 entrées supplémentaires, `.` et `..`. Le premier désigne le répertoire courant. Le second le répertoire parent. On peut se déplacer en utilisant soit un chemin *relatif* soit un chemin *absolu* avec la commande `cd`. On crée un répertoire avec la commande `mkdir`.

absolu

```
% cd /home/pham/nsmail
% pwd
/home/pham/nsmail
```

relatif

```
% cd ../../nsmail
% pwd
  /home/pham/nsmail
```

Manipuler les fichiers

cp

cp - copy file	cp *file1 file2* cp *file1 dir* cp *file1 file2 file3 dir*

```
% cp myMail.txt myMail.save
```

mv

mv - move file	mv *file1 file2* mv *file1 dir* mv *file1 file2 file3 dir*

```
% mv myMail.txt ../tmp/myMail.save
```

rm

rm - remove file	rm *file* rm -r *dir*

```
% rm myMail.txt
```

Création de liens

UNIX offre un mécanisme de liens très utile pour réaliser des alias sur les fichiers et les répertoires. Les cas d'utilisations sont par exemple: (i) on installe une nouvelle version d'un logiciel, tout en conservant l'ancienne, on peut créer un alias qui lancerait la nouvelle version avec le nom de l'ancien, et renommer l'ancien. (ii) éviter des chemins d'accès trop compliqués en créant un lien sur un répertoire donné.

ln [-s] orig dest

`ln` permet de créer des liens, l'option `-s` crée des liens symboliques. C'est généralement le type de liens qu'on utilise le plus.

```
% ln -s f1 f1Alias
% ln -s $HOME/bin bin
% ls -l
  lrwxrwxrwx 1 pham  user_  14 Oct  2 19:03 bin -> /h1/pham/bin/
  -rwxrwxrwx 1 pham  user_  13 Oct  2 19:03 f1
  lrwxrwxrwx 1 pham  user_  13 Oct  2 19:03 f1alias -> f1
```

Un lien dur crée une nouvelle entrée au répertoire avec le numéro d'i-node pointant sur celui du fichier original. C'est une association plus stricte (le fichier original doit exister!).

Afficher le contenu d'un fichier

cat

cat - concatenate file | cat *file1*

```
% cat /etc/motd
Hello, welcome...
```

more and less

more, less - page par page | more *file*
less *file*

```
% more Hello.txt
Hello World...
```

head and tail

head, tail - first, last part | head -n N *file*
tail -n N *file*

```
% head -n 2 adresse.dat
  Steve    3435 Boelter Hall  (310) 825-5657
  Claudia  3231 Boelter Hall  (310) 005-6575
% tail -n 1 adresse.dat
  pham     3231 Boelter Hall  (310) 825 4885
```

Editer un fichier: vi

`vi` (vee eye) est un éditeur simple qui a l'avantage d'être présent dans tous les systèmes UNIX. L'interface utilisateur est cependant rebutante. Pour des modifications légères, `vi` peut être plus pratique car il se charge vite.

`vi` possède 2 modes: un mode commande et un mode entrée de texte. Au lancement, `vi` est en mode commande. Pour passer en mode entrée de texte, taper i. Pour repasser en mode commande, taper ESC. Voilà ce que l'on doit voir lorsque l'on lance `vi sample`, qu'on tape i et qu'on rentre le texte `premier pas en vi`.

```
premier pas en vi
\~
\~
\~
\~
‘‘sample’’ [New file]
```

Si tout va mal, ESC :q!

Sauver ESC :w

Effacerligne `dd`

Editer un fichier: `emacs`

`emacs` est un éditeur fournit par GNU qui peut fonctionner sous X Window. Les commandes peuvent être appelées avec la séquence `M-x` où `M` est la touche `méta`. Avec un clavier PC, c'est souvent la touche alt. On peut aussi utiliser les séquences avec la touche ctrl, notée `C`. Par exemple, pour sauver un fichier on ferait `C-x C-s`.

`C-x C-f`	ouvre un fichier
`C-x C-s`	sauve un fichier
`C-x C-c`	quitte `emacs`
`C-x C-w`	sauve le fichier sous un autre nom
`C-w`	coupe une région
`C-y`	colle
`C-k`	supprime jusqu'à la fin de la ligne
`C-s`	recherche incrémentale
`C-x r k`	coupe une région rectangulaire
`C-x r y`	colle une région rectangulaire
`ESC-q`	formatte le texte

`M-x query-replace`	recherche et remplace - intéractif
`M-x replace-string`	recherche et remplace
`M-x string-rectangle`	insère une chaîne au début d'une région rectangulaire

Le(s) script(s) de login

A chaque login d'utilisateur, le shell lit un certain nombre de fichiers avant de donner la main à l'utilisateur.

Bourne-Shell

1. `/etc/profile`
2. `.profile`

Korn-Shell

1. `/etc/profile`
2. `.profile`

C-Shell

1. `/etc/csh.login` ou parfois `/etc/csh.cshrc`
2. `.cshrc`
3. `.login`, si le shell est celui de login.

Les scripts dans `/etc` effectuent les initialisations pour tous les utilisateurs, ceux locaux à chaque utilisateur permettent à celui-ci de personnaliser son environnement.

$HOME/.cshrc et $HOME/.login

Après la lecture des fichiers de configuration système, le C-shell tente de lire des fichiers spécifiques définis localement chez l'utilisateur.

.cshrc

En C-shell, le fichier local `.cshrc` est exécuté à chaque fois qu'une shell est lancé. Ce fichier sert généralement à initialiser les variables shell (path), les alias, les commandes relatives à la gestion de l'historique...

.login

Le fichier local `.login` est ensuite lu, mais uniquement au login, et sert à initialiser les variables d'environnement par exemple. Toutes les commandes que l'utilisateur désire lancer au démarrage d'une session, et non pas à chaque fois qu'un shell est lancé, devraient être mises dans le fichier `.login`.

Les variables

Le shell, et d'autres programmes, utilisent un certain nombre de variables. Il y a celles interne du shell et celles dites d'environnement.

Les variables du shell n'ont qu'une portée limitée au shell actif. Pour affecter une valeur à une variable du C-shell, on utilise la commande `set` avec le caractère =. En `sh`, il n'y a pas de `set`.

```
% set prompt='[\!] ready% '
[23] ready%
```

Pour afficher la liste des variables du shell, on utilise la commande `set` sans arguments.

```
[23] ready% set
history 15
home    /home/pham
path    (/usr/local/bin /bin /usr/bin)
prompt  [!] ready%
shell   /bin/csh
uid     5230
user    pham
[24] ready%
```

Les variables d'environnement

Les variables d'environnement sont accessibles à d'autres programmes par un mécanisme d'exportation. En C-shell, on utilise la commande `setenv` pour définir la valeur d'une variable d'environnement. En `sh` on fait `export` *variable*.

On utilise la commande `printenv` ou `env` pour afficher la liste des variables d'environnement. On a l'habitude de noter les variables du shell en minuscule et les variables d'environnement en majuscule.

```
% setenv MA_VARIABLE ‘‘Ceci est ma variable’’
% env
HOME=/home/pham
SHELL=/bin/tcsh
USER=pham
GROUP=user_
PWD=/home/pham/tex
MA_VARIABLE=ceci est ma variable
```

Pour référencer une variable, on utilise la caractère d'expansion **$**.

```
% echo MA_VARIABLE
MA_VARIABLE
% echo $MA_VARIABLE
ceci est ma variable
```

Intéractions avec le shell

Le shell interprète la ligne d'entrée comme des invocations de commandes. Certaines commandes sont internes au shell, comme `cd` et `exit`, d'autres sont externe, comme `ls` et `cp`.

Pour exécuter plusieurs commandes les unes après les autres, on utilise le caractère ;

```
% cmd1; cmd2; cmd3 ...
```

Parfois, l'exécution d'un programme est conditionnée par le résultat d'un autre. On peut utiliser les caractères && et || pour exprimer les conditions d'exécution.

&&	exécute la commande seulement si la commande précédente s'est correctement terminée.
||	exécute la commande seulement si la commande précédente s'est mal terminée.

Exemple

```
% ls /etc/shadow && echo ``Secure system''
% ls /etc/shadow || echo ``Ohh, bad''
Ohh, bad
```

Comment le shell trouve t-il les commandes?

Le shell cherche les commandes externes en utilisant une liste de chemin d'accès bien définie. Cette liste est représentée par la variable shell `$path`. Généralement, il y aura également une variable d'environnement `$PATH` exportée.

```
% echo $path
. /usr/local/bin /bin /usr/bin /usr/X11R6/bin /sbin
/usr/sbin
% set path = ($path $HOME/bin)
% echo $path
. /usr/local/bin /bin /usr/bin /usr/X11R6/bin /sbin
/usr/sbin /home/pham/bin
```

L'ordre de recherche va de la gauche vers la droite. Si le shell ne trouve pas la commande dans les chemins spécifiés, le message `Command not found` sera affiché.

Noter que le path inclut aussi  pour chercher en premier lieu dans le répertoire courant. Cette recherche n'est pas effectuée par défaut!

Les alias

On peut lancer des commandes qui ne possède pas un exécutable du même nom en créant un *alias* avec la commande `alias` du shell.

```
% alias ll ls -l
% alias dir ll -a
% ll
drwxr-xr-x   2 pham   user_   1024 Sep 18 10:36 doc/
drwxr-xr-x   2 pham   user_   1024 Sep 20 16:34 figure/
% dir
drwxr-xr-x   5 pham   user_   1024 Sep 20 19:19 ./
drwxr-xr-x   9 pham   user_   1024 Sep 15 18:00 ../
drwxr-xr-x   2 pham   user_   1024 Sep 18 10:36 doc/
drwxr-xr-x   2 pham   user_   1024 Sep 20 16:34 figure/
```

unalias

On peut enlever un alias en utilisant `unalias`. L'effet dure jusqu'à la prochaine utilisation de `alias` pour le même alias.

```
% alias rm rm -i
% rm hello
rm: remove ‘hello’? n
% unalias rm
```

On peut faire référence à la commande sans l'alias en préfixant la commande par \.

```
% \rm hello
% ls hello
ls: hello: No such file or directory
```

L'historique

Le C-shell garde en mémoire un certain nombre de commandes précédentes qui constituent un historique.

```
% set history = 15
% set savehist = 10
% set prompt = ``[\!] % ''
```

history

```
[3] % history
      3  history
      2  cd TP
      1  pwd
```

On peut référencer une ancienne commande par sa position absolue avec le caractère `!`, sa position relative avec `!-` ou par son text avec `!`*text*. La dernière commande est référencée par `!!`.

```
[4] % !2
      cd TP
[5] % !-4
      pwd
      /home/pham/TP
[6] % !p
      pwd
```

Réutiliser des arguments d'anciennes commandes

On peut référencer les arguments d'une ancienne commande avec le caractère : suivi de la position de l'argument. L'argument 0 est le nom de la commande. Le dernier argument est référencé par $, le second par ^. Un interval peut être référencé par -. Pour indiquer tous les arguments on utilise *.

```
cmd1 arg1 arg2 arg3 | cmd2 arg1 arg2 &

 0    1    2    3   4   5   6    7   8
      ^                              $
      * ------------------------------>
```

```
[1] % cp a1.c a2.c a3.c tmp
[2] % echo !1:$
      tmp
[3] % cp !1:^ a1.c.save
[4] % rm !1:^-3
[5] % mv !1:*
```

Que donnerait `% !1:0 !3:* !4:$ !5:4`

L'historique - résumé

`!!`	répète la dernière commande
`!n`	répète la commande *n*
`!-n`	répète la *n-ième* dernière commande
`!str`	répète la première commande commençant par *str*
`!?str?`	répète la première commande contenant *str*
`!:`	répète la dernière commande, avec un modificateur
`!:n`	sélectionne le *n-ième* argument de la dernière commande
`!:n-m`	sélectionne les arguments, du *n-ième* au *m-ième* de la dernière commande
`!^`	sélectionne le premier argument de la dernière commande
`!*`	sélectionne tous les arguments de la dernière commande
`!:n*`	sélectionne les arguments, du *n-ième* au dernier, de la dernière commande
`!:n-`	sélectionne les arguments, du *n-ième* à l'avant dernier, de la dernière commande
`^`str_1`^`str_2	remplace str_1 par str_2 dans la dernière commande
`!n:s/`str_1`/`str_2`/`	remplace str_1 par str_2 dans la *n-ième* commande

Remarque

la commande `alias` utilise la même syntaxe que celle de l'historique pour fournir des arguments aux alias.

La gestion des processus

Un processus sous UNIX peut être qualifié de `Runnable`, `Sleeping`, `Stopped` et `Zombie`.

UNIX est multi-tâches. Par défaut un processus est lancé en avant-plan. On peut lancer un processus en arrière plan en suffixant le nom de l'exécutable avec le caractère `&`.

```
% ls &
[4] 1291
% Archive_2   Cours    bin    cdrom    nsmail
tmp
[4]    Done                          ls
%
```

Le nombre entre crochets est le `job number`, différent du `Process IDentifier` (PID).

Un processus lancé en avant-plan peut être interrompu en utilisant `ctrl+z`.

Afficher les processus

ps

process status: `ps -[option]`

Sans option, `ps` affiche les processus de l'utilisateur.

```
% ps
  PID TTY STAT  TIME COMMAND
  299   1 S    0:00 /bin/login -- pham
  307   1 S    0:00 -tcsh
  996  p0 R    0:00 ps
```

top

display top CPU processes: `top -[option]`

Sans option, `top` affiche la liste des processus et la proportion de temps CPU qu'ils utilisent. Taper q pour quitter.

```
% top
  6:45pm  up  6:12,  2 users,  load average: 0.01, 0.06, 0.05
  35 processes: 34 sleeping, 1 running, 0 zombie, 0 stopped
  CPU states:  2.3% user,  3.1% system,  0.0% nice, 94.5% idle
  Mem:   30844K av,  30256K used,    588K free,  27028K shrd,    116K buff
  Swap:  68540K av,     16K used,  68524K free                 13084K cached

  PID USER     PRI  NI  SIZE  RSS SHARE STAT  LIB %CPU %MEM   TIME COMMAND
 1289 pham      14   0   584  584   456 R       0  5.5  1.8   0:00 top
    1 root       0   0   404  404   340 S       0  0.0  1.3   0:03 init
    2 root       0   0     0    0     0 SW      0  0.0  0.0   0:00 kflushd
    3 root     -12 -12     0    0     0 SW<     0  0.0  0.0   0:00 kswapd
  307 pham       0   0   780  780   560 S       0  0.0  2.5   0:00 tcsh
  299 pham       0   0   924  924   560 S       0  0.0  2.9   0:00 login
```

Manipuler les processus

bg fg

Un programme lancé en avant-plan et interrompu avec ctrl+z peut être ensuite mis en arrière plan avec bg. Il peut également être remis en avant-plan avec `fg`.

```
% ghostview intro.ps   -->  ctrl+z
Suspended
% bg
[4]    ghostview intro.ps &
% fg %4
```

kill terminate a process: `kill -[signal]` *pid*

`kill` sert à envoyer des signaux aux processus. Par défaut, le signal `TERM` est envoyé. Si le processus attrape ce signal, il peut être nécessaire d'envoyer le signal `KILL` (9) pour pouvoir tuer le processus.

```
% ghostview intro.ps &
[4]  1254
% kill 1254
[4]    Terminated          ghostview intro.ps
%
```

Les métacaractères

Le shell réserve un certain nombre de caractères pour indiquer des actions spéciales. Ces métacaractères sont classés en 5 catégories: pour la syntaxe, pour les fichiers, pour la quotation, pour les E/S et pour l'expansion/substitution.

syntaxe

;	sépare les commandes
|	pipes
( )	isolent les commandes
&	lance en tâche de fond
||	séparateur conditionnel, *si échoue*
&&	séparateur conditionnel, *si succès*

fichiers

?	remplace un caractère
*	remplace plusieurs caractères
[ ]	indique un intervalle - `ls essai.[a-z]`
{ }	indique des alternatives - `ls l{aio}st`
~	remplace la racine pour l'utilisateur
/	délimite les composantes d'un chemin

Les métacaractères - suite

quotation

\	évite l'interprétation d'un métacaractère
' '	prend une chaîne de caractères littéralement - single quote
" "	évite l'interprétation des métacaractères dans une chaîne mais permet l'expansion et l'évaluation - double quote
` `	évalue la chaîne de caractères - back quote

Examples:

```
% echo \*
  *
% echo 'hd/?d:dl$kd/*?'
  hd/?d:dl$kd/*?
% echo ``*/*/* $shell''
  ...
% echo ``Hello from'' `pwd` ``on'' 'hostname'
  ...
```

expansion/substitution

$	substitution de variable – echo $PATH
!	substitution d'une commande de l'historique

Le problèmes des métacaractères

Les métacaractères **!,** \ **et '** ne peuvent pas être pris litéralement avec l'utilisation des ' '. On utilise pour cela le caractère \.

Exercise

En utilisant la commande `echo`, afficher les chaînes de caractères '!\ et '*.

Les outils les plus courants

L'utilisateur novice est souvent désorienté par la multitudes commandes disponibles et par le nombre d'options de ces commandes.

Cependant, cette multitude constitue toute la base d'UNIX et une fois qu'un certain nombre de commandes sont devenues familières, on prend vite conscience qu'il est possible de faire presque tout avec UNIX.

tar, (un)compress, g(un)zip

tar

Archive | tar cvf *tar_file* [*file*...]
tar xvf *tar_file* [*file*...] -C dir

```
% tar cvf backup.tar Cours/ Papers/WSC.tex
% tar xvf backup.tar -C tmp
```

(un)compress

(de)compression | (un)compress [*file*...]

```
% compress Paper.ps
% ls
Paper.ps.Z
% uncompress Paper.ps.Z
```

g(un)zip

(un)zip fichiers | g(un)zip -[option] [*file*...]

```
% gzip Paper.ps
% ls
Paper.ps.gz
```

`grep` et les expressions régulières

grep

Recherche d'expression | grep *expression file*...

```
% grep '#define HLA' FedAmb.c rti.c
  FedAmb.c:#define HLA  /* definition */
  rti.c:#define HLA /* encore une */
```

`grep` est un outils puissant permettant l'utilisation d'expressions régulières comme motif de recherche. Il affiche une ligne si une partie de celle-ci satisfait l'expression donnée en paramètre.

`.`	reconnaît tout caractère
`*`	reconnaît 0 ou plusieurs occurences du caractère précéden
`[abc]`	reconnaît tout caractère dans {abc}
`[a-d]`	reconnaît tout caractère dans l'intervalle [a-d]
`[^exp]`	reconnaît tout caractère non inclus dans l'expression
`^abc`	abc doit commencer une ligne
`\{n,m\}`	reconnaît l'expression précédente pour un minimum de `n` fois et un maximum de `m` fois.
`\ <abc\ >`	reconnaît abc si c'est un mot
`\(abc)\`	sauve abc dans un buffer. 9 buffers peuvent être définis.
`\n`	reconnaît la *n-ième* expression sauvegardée

`find` et `which`

find

Recherche de fichiers | find *dir* -[search options] [action]

```
% find . -name ``*.c'' -print
  Program/toto.c
  Divers/rand.c
% find . -name ``*.log'' -print -exec rm {} \;
  tex/Paper/PADS99.log
  tex/Paper/WSC98.log
  tmp/dump.log
```

Il semble bizarre que find n'affiche rien par défaut. En fait `find` a été conçu pour fonctionner avec `cron`.

which

Chemin complet de la commande | which *cmd*

```
% which cp
  /bin/cp
```

sort, cut

sort

Tri de lignes | sort -[option] [+pos] [-pos] *file*...

```
% sort /etc/passwd
% sort +1 -3 database.txt
```

`sort` numérote les colonnes à partir de 0. `+1` indique d'utiliser comme clé la colonne 1. `-3` indique que la clé est constituée des colonnes 1 et 2, sinon c'est le reste de la ligne.

cut

Supprime des parties de lignes | cut -f *champs file*...

```
% cat bd.txt
  Paul    23ans   12 rue de la grange aux belle   Marié
  Sophie  22ans   34 rue berthelot                Célibataire
% cut -f 1,2 bd.txt
  Paul    23ans
  Sophie  22ans
% cut -f 1,2-4 bd.txt
  Paul    23ans   12 rue de la grange aux belle   Marié
  Sophie  22ans   34 rue berthelot                Célibataire
```

Attention: `cut` utilise les tabulations comme délimiteurs par défaut.

tr, paste

tr

`tr` permet de substituer des caractères. La forme la plus simple `tr str1 str2 < file` substitue caractère par caractère les caractères de `str1` par les caractères à la même position dans `str2`.

On peut traduire des minuscules en majuscules avec `tr '[a-z]' '[A-Z]'`. On peut aussi enlever tous les caractères dans un ensemble donné avec `tr -d ''aeiouy'' < file1 > file2`.

paste

`paste` permet de coller une colonne dans un fichier.

```
nom:            date:
    jean             3/10/70
    marc             5/8/71
    alice            12/5/69
% paste nom date > base.txt
% cat base.txt
  jean    3/10/70
  marc    5/8/71
  alice   12/5/69
```

sed, awk

sed

Filtre | sed *expression file*...

```
% sed y/str1/str2/       - pareil que tr str1 str2
% sed s/str1/str2/       - remplace str1 par str2
```

awk

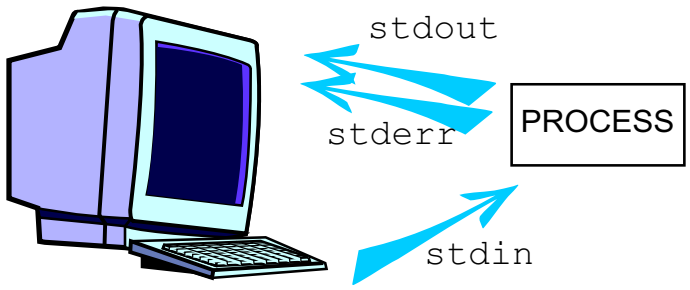
Bon a tout faire... | awk [-f prog_file] [cmd] *file*...

```
data.txt:
          jean    51
          marie   19
% awk '{print $2; print$1}' data.txt
  51
  jean
  19
  marie
% awk '{x+=$2} END {print ``sum=''x}' data.txt
  sum=70
% awk '{print NR``\t''$0}' data.txt
  1     jean    51
  2     marie   19
```

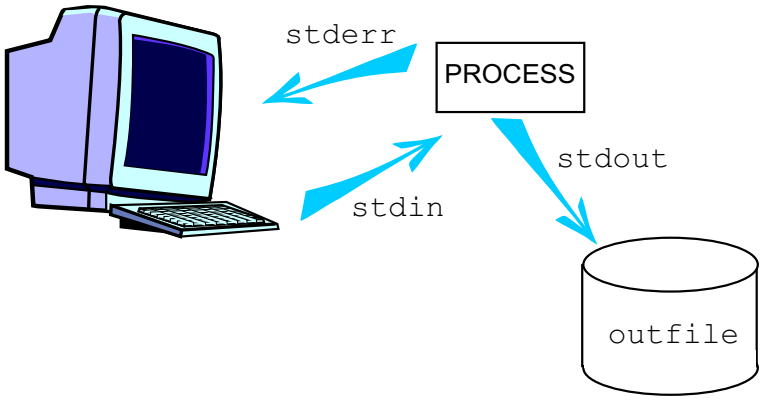
Les redirections

UNIX ouvre pour chaque processus 3 fichiers appelés `stdin`, `stdout` et `stderr`. Ils ont pour descripteurs 0,1 et 2 respectivement.

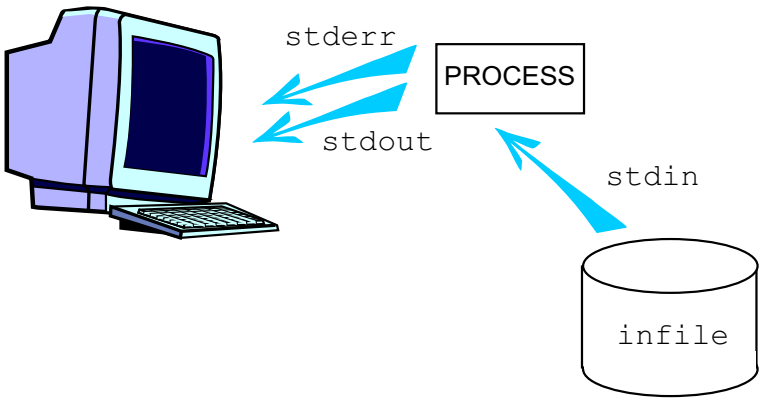
Par défaut



`command` > *outfile*



`command` < *infile*



Les redirections, suite...

Rediriger stdout en ajoutant à la fin

```
% command >> outfile
```

Rediriger stdin et stdout

```
% command < infile > outfile
```

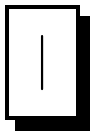
Rediriger stderr

```
$ command 2> errorfile
% command >& errorfile
```

Rediriger stderr et stdout

```
$ command > outfile 2> errorfile
% (command > outfile) >& errorfile
```

Les `pipes`

UNIX offre la possibilité d'utiliser la sortie d'une commande `a` comme entrée d'une autre commande `b`, i.e. utiliser `stdout` de `a` comme `stdin` de `b`. Ce mécanisme est appellé *pipe* et s'effectue en utilisant le caractère  .

```
% cmd1 | cmd2 | cmd3 | ...
```

Par exemple,

```
% who > who_is_on
% wc -l < who_is_on
  3
```

devient,

```
% who | wc -l
  3
```

Ce mécanisme est très important dans UNIX. Avec la philosophie de base d'UNIX: *"beaucoup de programmes simples"*, les `pipes` permettent de combiner les actions de simples programmes pour obtenir une action complexe.

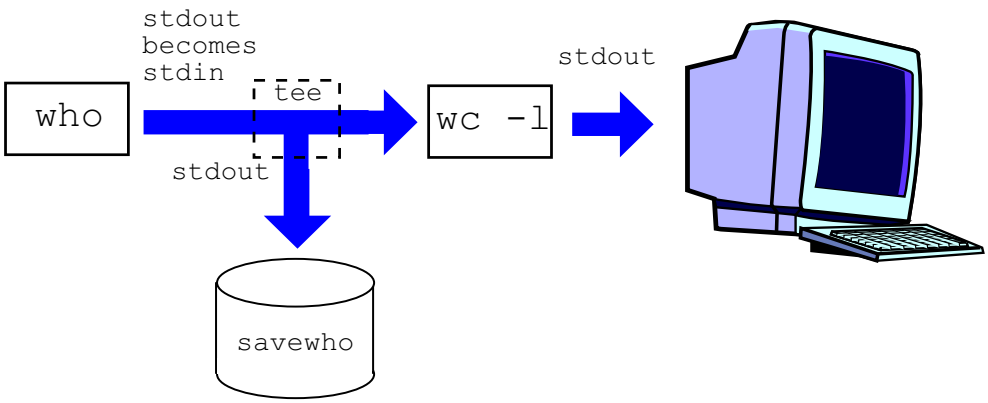
La commande `tee`

Utiliser `stdout` de `a` pour `b` et en même temps le rediriger vers un fichier s'effectue avec la commande `tee`.

`tee`

`tee` permet d'écrire un flot de donnée vers un ficher sans changer la manière dont les pipes sont utilisés.

```
% who | tee savewho | wc -l
3
% cat savewho
  pham      tty1      Sep 20 16:32
  pham      ttyp0     Sep 20 16:32 (:0.0)
  steve     ttyp4     Sep 19 14:03
```



Programmer avec le shell - les scripts

Une des caractéristiques principales d'un shell est la possibilité d'écrire des scripts qui seront exécutés comme des commandes. Dans UNIX, un grand nombre de commandes sont en fait des scripts écrits le plus souvent en `sh`.

Un script est un fichier texte qui contient une succession de commandes du shell. Pour qu'un script soit exécutable, il fait qu'il ait la permission à l'exécution (`chmod +x`). Les commentaires commence par #.

Un script peu être lancé à la ligne de commande, auquel cas il s'exécute avec le shell courant. On peut également indiquer explicitement le shell à utiliser en lançant le script comme argument du shell: `csh script`. Une solution plus élégante consiste à mettre une ligne spécial au début du script pour indiquer le shell nécessaire. Cette solution permet de lancer un script à partir de n'importe quel shell, le bon shell sera d'abord lancé pour interpréter le script.

```
#!/bin/csh
...
```

Les instructions du C-shell

Le C-shell possède un certain nombre d'instructions similaires au langage C. On aura par exemple:

```
if ( _exp ) then
   cmdlist
[else]
   ...
endif
```

```
while ( _exp )
   cmdlist
   [break]
   [continue]
end
```

```
switch ( _word )
  case _pattern:
             cmdlist
             ...
             breaksw
  default:
             cmdlist
             breaksw
endsw
```

```
foreach _var (_list)
   cmdlist
   ...
end
```

```
_loop:
  cmdlist
  ...
goto _loop
```

Les instructions de `sh`

`sh` possède aussi un certain nombre d'instructions. On aura par exemple:

```
if [ _exp ]
then
   cmdlist
[else]
   ...
fi
```

```
while [ _exp ]
do
   cmdlist
   [continue]
   [break]
done
```

```
case _param in
  _pattern) cmd;;
            ...
  *) cmd;;
     ...
esac
```

```
for _var in _list
do
  cmdlist
done
```

```
until _cond
do
  cmdlist
done
```

Test sur les fichiers

Le C-shell offre les commandes suivantes pour effectuer divers tests sur les fichiers. Elles s'utilisent avec les instructions de test telles que `if` et `while` par exemple.

`-d`	test si le fichier est un répertoire
`-e`	test si le fichier existe
`-f`	test si le fichier est un fichier normal
`-o`	test si le fichier m'appartient
`-r`	test si je peux lire le fichier
`-w`	test si je peux écrire dans le fichier
`-x`	test si je peux exécuter le fichier
`-z`	test si la taille du fichier est nulle

exemple

```
#!/bin/csh

if (-e .display) then
   setenv DISPLAY `cat .display`
   echo ``DISPLAY set to default host''
endif
```

Manipulation des variables

Variables caractère

`set` sert à affecter une valeur à une variable caractère. Cette variable peut ensuite être référencer en utilisant `$`.

```
whereami:
        set msg = ‘‘Bonjour vous êtes sur ’’
        set hn  = ‘hostname‘
        echo $msg $hn
$ whereami
  Bonjour vous êtes sur lhpca.univ-lyon1.fr
```

Variables numériques

Pour les variables numériques, on utilise le caractère `@` pour affecter une valeur numérique, mais toujours `$` pour la référencer.

```
calcul:
        @ h = 6
        @ j = 7
        @ h = $j + 6
        echo $h
% calcul
  13
```

Les opérateurs

csh	sh	actions
( )	( )	Modifie l'ordre des évaluations
+		Addition
-		Soustraction
*		Multiplication
/		Division
%		Modulo
^		Opérateur OR bit exclusif
~		Complément à 1
==	=	Comparaison chaîne de caractère
!=	!=	Non égal
!	!	Négation
<	-lt	Inférieur à – (operand␣<␣operand)
>	-gt	Supérieur à
>=	-ge	Supérieur ou égal à
<=	-le	Inférieur ou égal à
>>		Décalage droite
<<		Décalage gauche
&		Opérateur AND bit
|		Opérateur OR bit inclusif
&&	-a	AND logique
||	-o	OR logique
=	=	Affectation
+=		raccourci pour x = x + y
-=		raccourci pour x = x - y
*=		raccourci pour x = x * y
/=		raccourci pour x = x / y
%=		raccourci pour x = x % y
^=		raccourci pour x = x ^ y
++		raccourci pour x = x + 1
--		raccourci pour x = x - 1

Pour ++ et --, il n'est pas nécessaire de préfixer la variable avec $ $\longrightarrow$ i++ et pas $i++.

Accéder aux arguments

Les arguments passés à un script peuvent être référencés en utilisant le caractère $ suivi d'une position. Le nom de la commande est l'argument 0.

```
% cmd arg1 arg2 arg3 arg4 ... arg9 arg10 arg11 ...

   $0  $1   $2   $3   $4       $9
```

Pour accéder aux arguments supérieurs à 9, il faut utiliser la commande `shift` qui décale les arguments vers la gauche, ainsi $2 devient $1.

```
cmd1:
      echo $1
      shift
      echo $1 $2

% cmd1 hello again hi
  hello
  again hi
```

Ecrire un script qui permet de demanderait à l'utilisateur son nom et prénom si ceux-ci n'ont pas été donnés en paramètre. Le script afficherait alors `Bonjour nom prenom, tu es sur` *machine*.

Variables prédéfinies C-shell

`$$`	numéro du processus
`$n`	$n \in [1, 9]$, arguments de la ligne de commande
`$*`	tous les arguments de la ligne de commande
`$argv[n]`	*n-ième* mot de la ligne de commande
`${argv[n]}`	*identique au précédent*
`$#argv`	nombre de mots dans la ligne de commande
`$?param`	retourne 1 si *param* est initialisé, 0 sinon

Exemples

```
test:
      echo '$$:'$$
      echo '$3:'$3
      echo '$0:'$0
      echo '$*:'$*
      echo '$argv[2]:'$argv[2]
      echo '${argv[4]}:'${argv[4]}
      echo '$#argv:'$#argv
      echo '$?nondef:'$?nondef

% test one two three four five
  $$: 1423
  $3: three
  $0: test
  $*: one two three four five
  $argv[2]: two
  ${argv[4]}: four
  $#argv: 5
  $?nondef: 0
```

Fournir des commandes/données au script

$< read

On peut lire une entrée à partir de l'entrée standard avec $< ou avec read en sh.

```
test_input:
            echo -n ‘‘Veuillez entrer votre nom: ‘‘
            set nom = $<
            echo ‘‘Bonjour $nom’’
```

$<< ident \ldots ident$

On peut également fournir des données à partir du script lui-même en utilisant les séquences $<< ident...ident$ pour délimiter la zone de donnée.

```
ecrire:
        cat greetings.txt > hello << EOF
        Hello, bienvenue
        EOF
% ecrire
% cat greetings.txt
  Hello, bienvenue
```

Utiliser la commande `source`

La commande interne `source` du C-shell permet de charger un script dans l'environnement actuel du shell. On l'utilise généralement lorsque l'on veut réactualiser la valeur des variables définies dans le fichier `.cshrc`.

```
% source .cshrc
% source myNewEnvironment
```

Remarque

Parfois, un script s'exécute mal s'il n'y a pas les lignes `#!/bin/csh` au début du script. On peut souvent utiliser `source` pour éviter de mettre ces lignes.

```
test:
      echo -n "Veuillez entrer votre nom: "
      set nom = $<
      echo "Bonjour $nom"
% test
  ./test: syntax error near unexpected token `$<'
% source test
  Veuillez entrer votre nom:
```

Résumé des différents shells

	sh	csh	ksh	bash	tcsh
Job control	N	Y	Y	Y	Y
Aliases	N	Y	Y	Y	Y
Shell functions	Y	N	Y	Y	N
"Sensible" Input/Output redirection	Y	N	Y	Y	N
Directory stack	N	Y	Y	Y	Y
Command history	N	Y	Y	Y	Y
Command line editing	N	N	Y	Y	Y
Vi Command line editing	N	N	Y	Y	Y
Emacs Command line editing	N	N	Y	Y	Y
Rebindable Command line editing	N	N	N	Y	Y
User name look up	N	Y	Y	Y	Y
Login/Logout watching	N	N	N	N	Y
Filename completion	N	Y	Y	Y	Y
Username completion	N	Y	Y	Y	Y
Hostname completion	N	Y	Y	Y	Y
History completion	N	N	N	Y	Y
Fully programmable Completion	N	N	N	N	Y
Co Processes	N	N	Y	N	N
Builtin artithmetic evaluation	N	Y	Y	Y	Y
Can follow symbolic links invisibly	N	N	Y	Y	Y
Periodic command execution	N	N	N	N	Y
Custom Prompt (easily)	N	N	Y	Y	Y
Spelling Correction	N	N	N	N	Y
Process Substitution	N	N	N	Y	N
Underlying Syntax	sh	csh	sh	sh	csh
Freely Available	N	N	N	Y	Y
Checks Mailbox	N	Y	Y	Y	Y
Tty Sanity Checking	N	N	N	N	Y
Can cope with large argument lists	Y	N	Y	Y	Y
Has non-interactive startup file	N	Y	Y	Y	Y
Has non-login startup file	N	Y	Y	Y	Y
Can avoid user startup files	N	Y	N	Y	N
Can specify startup file	N	N	Y	Y	N
List Variables	N	Y	Y	N	Y
Full signal trap handling	Y	N	Y	Y	N
File no clobber ability	N	Y	Y	Y	Y
Local variables	N	N	Y	Y	N

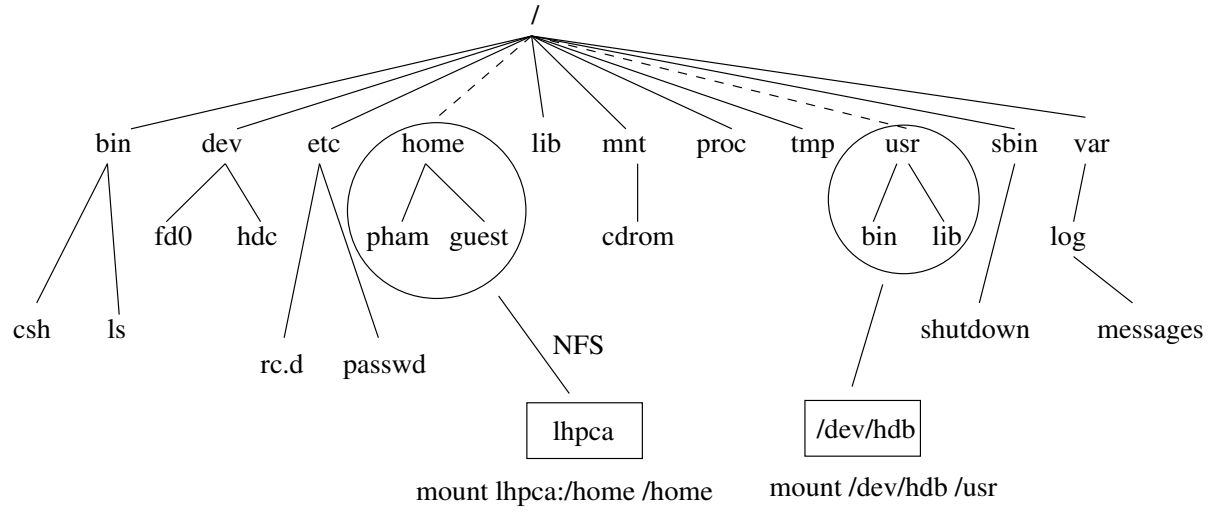
Partitionnement, montage, NFS

Un disque dur est généralement divisé en plusieurs partitions pour d'une part établir un cloisonnement logiques des données et d'autre part pour réduire les risques de corruption.

Un système peut également avoir plusieurs disques. Chacun avec ses propres partitions et hiérarchies de répertoire.

UNIX possède un mécanisme de montage qui permet d'offrir la vision d'un système de fichier homogène à partir de différents disques et partitions.

Une extension au montage via le réseau à été crée par SUN et s'appelle NFS pour Network File System. NFS permet de monter une hiérarchie de répertoire se trouvant sur une machine distante, via les services de réseaux.



Utiliser le réseaux

Le modèle TCP/IP est utilisé sur toutes les machines qui composent l'Internet. Ce modèle utilise une adresse sur 32 bits pour identifier de manière unique une machine connectée à l'Internet. On utilise couramment une forme constituée de 4 nombres entre 0 et 255 séparés par des points. Ensuite, on associe une chaîne de caractères à cette adresse comme mnémonique.

```
10010011.11000011.01100000.01010001   (binaire)
  8  3  .  B  3  .  6  0  .  5  1      (hexa)
         131.179.96.81                 (décimal)
       badal.cs.ucla.edu               (caractère)
```

L'utilisateur utilise généralement la forme caractère, des tables peuvent transformer cette chaîne en adresse réelle pour la machine. Toutes les machines ne sont pas connectées directement à l'Internet, certaines machines servent de passerelles pour connecter un sous-réseau vers l'extérieur.

Une adresse comme `badal.cs.ucla.edu` signifie la machine nommée `badal` dans le domaine `cs.ucla.edu`.

nslookup et ping

nslookup

```
% nslookup
  > peg.cs.ucla.edu
    peg.cs.ucla.edu     A    131.179.96.134
```

`nslookup` permet d'interroger de manière interactive les serveurs de noms pour obtenir l'adresse IP correspondant au nom de la machine.

ping [hostname] [IPaddr]

```
% ping lynx.cs.ucla.edu
  PING lynx.cs.ucla.edu (131.179.160.60): 56 data bytes
  64 bytes from 131.179.160.60: icmp_seq=0 ttl=235 time=507.8 ms
  64 bytes from 131.179.160.60: icmp_seq=1 ttl=235 time=525.6 ms
  64 bytes from 131.179.160.60: icmp_seq=2 ttl=235 time=661.5 ms
  64 bytes from 131.179.160.60: icmp_seq=3 ttl=235 time=414.0 ms

  --- lynx.cs.ucla.edu ping statistics ---
  4 packets transmitted, 4 packets received, 0% packet loss
  round-trip minåvg/max = 414.0/527.2/661.5 ms
```

`ping` envoie des paquets ICMP vers la machine cible et mesure le temps aller-retour. On se sert de `ping` pour savoir si une machine est accessible.

ftp, telnet, rlogin

ftp

File Transfer Protocol

```
% ftp peg.cs.ucla.edu
Connected to 131.179.96.134
Name: pham
Passwd:
Connected.
ftp>
```

les commandes les plus courantes sont: `cd`, `lcd`, `put`, `get`, `bin` et `help`.

telnet

`telnet` permet de se connecter à une machine distance avec le protocol TELNET. Identification requise.

rlogin

`rlogin` permet à peu près la même chose que `telnet`. Il permet cependant de passer l'environnement de l'utilisateur à la machine hôte. A terme, il semblerait que `rlogin` soit remplacer par `telnet`. Un fichier `.rhosts` permet d'indiquer les machine depuis lesquelles l'identification n'est pas requise.

L'interface graphique - X Window

Les systèmes UNIX sont généralement livrés avec une interface graphique nommée X Window. La version actuelle est X Window v11 (notée `X11`). X Window ne définit que les fenêtres, pas comment elles ressemblent. On peut mettre au dessus de X11 une grande variété de *Window Manager* comme `fvwm`, `twm`...

Les programmes fonctionnant sous X11 commence généralement par la lettre `x`. Certains cependant sont de manière inhérente fenétrés, comme `netscape`, et il n'est pas nécessaire de les préfixer par `x`.

X11 peut être lancé automatiquement ou manuellement. Avec `xdm`, c'est X11 d'emblée. Dans ce cas, `xdm` cherche un fichier `$HOME/.xsession` pour exécuter les commandes initiales. Si on utilise le mode manuel, en lançant `xinit` ou `startx` ou `xstart`, c'est le fichier `$HOME/.xinitrc` qui est exécuté.

DISPLAY, xhost

La variable d'environnement `DISPLAY` indique la machine cible pour les redirections d'affichage. Par défaut, si cette variable n'est pas mise, c'est la machine sur laquelle on se logge qui est la machine cible. On peut indiquer une autre machine par la commande suivante:

```
% setenv DISPLAY peg.cs.ucla.edu:0.0
```

xhost

La machine cible doit cependant autoriser l'affichage en utilisant la commande `xhost`.

```
% xhost +lhpca.univ-lyon1.fr
% xhost -peg.cs.ucla.edu
% xhost +
```

Mail, web

On peut lire son mail avec `mail` mais il est plus facile et conviviale d'utiliser `netscape` pour cela. Il suffira de configurer `netscape` pour qu'il utilise le bon serveur de mail (voir TP). Une adresse mail se présente sous la forme suivante `nom@machine.domain`. Par exemple `cpham@lhpca.univ-lyon1.fr`.

Le WEB est une application basée sur les hypertextes et les hyperliens, originellement crée au CERN. Une adresse WEB s'écrit généralement sous la forme `http://www.amazon.com` mais on pourrait aussi utiliser une adresse IP `http://208.216.182.15`

Une page WEB se lit avec un *browser* comme `netscape`, `Microsoft Explorer` ou `Mosaic`. Le protocole utilisé est `http` qui permet l'interrogation et le transfert de pages WEB écrites en `HTML`, `JavaScript` ou `java`.

En général, comment obtenir de l'aide?

Sur un système UNIX, la plupart des commandes possèdent des manuels d'aide accessibles avec la commande `man`. On utilise le plus souvent `man` suivi du nom de la commande que l'on souhaite connaître.

```
% man find
  FIND(1L)                                                    FIND(1L)

  NAME
       find - search for files in a directory hierarchy

  SYNOPSIS
       find [path...] [expression]

  DESCRIPTION
       This  manual page documents the GNU version of find.  find
       searches the directory tree rooted at each given file name
       by  evaluating  the  given  expression from left to right,
       according to the rules of precedence (see  section  OPERA-
          . . .

  SEE ALSO
     locate(1L),  locatedb(5L), updatedb(1L), xargs(1L) Finding
     Files (on-line in Info, or printed)
```

Utiliser intensivement `man`!!