Unix- Linux : les bases indispensables
UX001

UX001 – V2.10 Show

Agenda théorique …

- Jour 1
  - Chap 1 - Introduction
    - Connexion
    - Documentation
    - Syntaxe
    - Premières commandes
      - TPs 1 & 2
  - Chap 2 - Systèmes de fichiers
    - File System
    - Types et noms des fichiers
    - Arborescence et répertoires
      - TP 3
    - Manipulations de fichiers
      - TP 4
    - Droits d'accès

- Jour 2
  - TP 5
  - Chap 3 - Editeur VI
    - Principes généraux
    - Quelques usages
      - TP 6
  - Chap 4 - Process
    - Attributs
    - Signaux
    - Redirections
    - « Pipe »
    - Arrière plan
      - TP 7

… agenda théorique.

- Jour 3
  - Chap 5 – Shell(s)
    - Historique
    - Variables
    - Caractères spéciaux
      - TP 8
    - Alias
    - Historique des commandes
    - Fichiers de connexions
      - TP 9
  - Chap 6 – Utilitaires
    - file, nl, cmp
    - find
    - tar, compression
    - Impressions
    - script, du, su, cut
    - sort, grep, …
- Jour 4
  - Chap 6 suite – Utilitaires suite
    - TP 10
  - Chap 7 – Réseau et X11
    - Résolution @IP
    - Quelques applications
    - Environnement graphique
  - Chap 8 – Bases programmation shell
    - Tests,
    - Boucles,
    - Case … esac
    - Fonctions,
    - Gestion des signaux
      - TP 11

Global Knowledge

Historique

1970 - Unics par K. Thompson & D. Ritchie chez ATT

1973 - Langage C → portabilité

Deux branches : ATT (System V) & Berkeley (BSD)

1983 - Projet GNU

1991 - OSF/1 + FreeBSD + Noyau Linux

Unix aujourd'hui

Types d'utilisateur	Utilisateurs non informaticiens	Utilisateurs informaticiens	Développeurs	Administrateurs Exploitants
Importance de la version	nul (contexte logiciel)	faible	faible	important

Linux

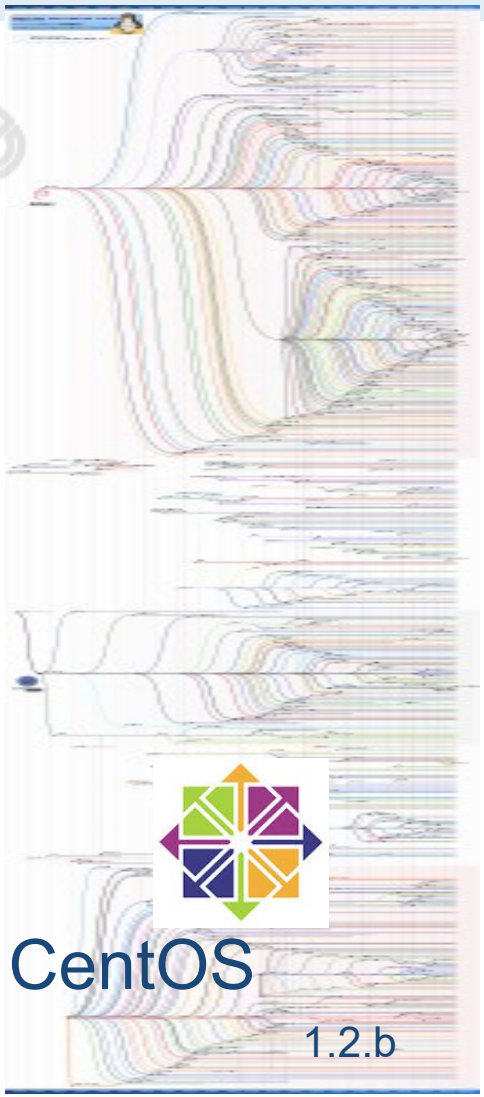
https://upload.wikimedia.org/wikipedia/commons/1/1b/Linux_Distribution_Timeline.svg

- Debian → Ubuntu

- slackware → SuSE

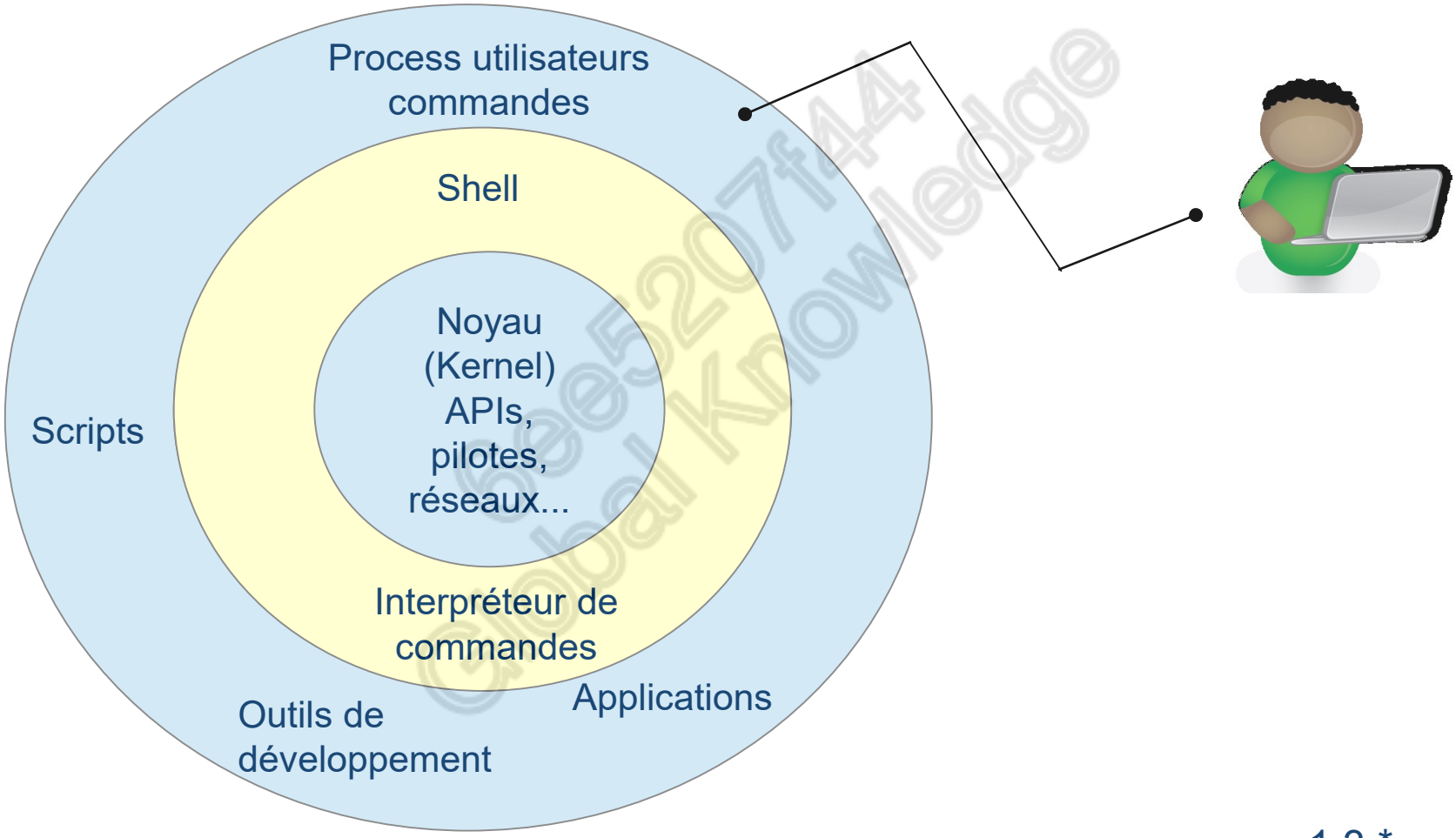
redhat

- RedHat → RedHat Enterprise Linux → CentOS



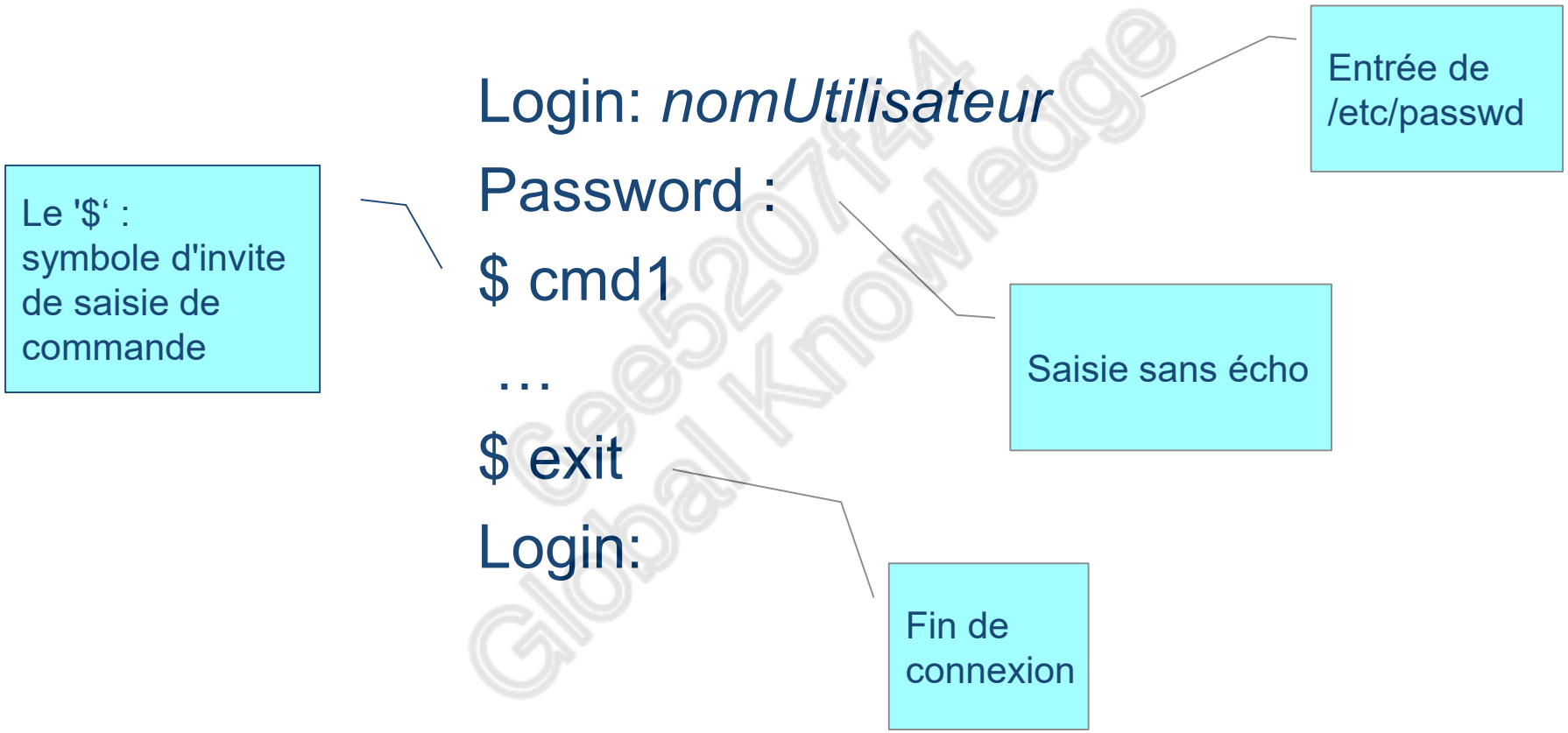
1.2.b

Composants d'un système Unix



1.3.*

Connexion / session utilisateur ...



Un utilisateur → une ligne de `/etc/passwd`

- Nom du compte
- Zone mot de passe historique
- Uid
- Gid à la connexion
- Commentaire
- Répertoire d'accueil
- Process lancé à la connexion → un shell

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
```


1.4.c

La documentation : `man`

Le manuel en ligne
aide sur le manuel lui même :

```
$ man man
User Commands                                                   man(1)
NAME
     man - find and display reference manual pages
SYNOPSIS
     man [-] [-adFlrt] [-M path] [-T macro-package]  [-s section]  name...
     man [-M path] -k keyword...
     man [-M path] -f file...
DESCRIPTION
The man command displays information from the reference manuals. It
displays complete manual pages that you select by name, or one-line
summaries selected either by keyword (-k), or by the name of an
associated file (-f). If no manual page is located, man prints an error
message.
.....................................................................
```


1.5.b

… man

```
$ man -k calendar
cal          cal(1)          display a calendar
calendar     calendar(1)     reminder service
difftime     difftime(3c)    computes the difference between two calendar
times
mktime       mktime(3c)      converts a tm structure to a calendar time
$
```

```
$ man passwd
User Commands                                              passwd(1)
NAME
     passwd - change login password and password attributes
.....................................................
```

```
$ man -s 4 passwd
File Formats                                               passwd(4)
NAME
     passwd - password file
SYNOPSIS
     /etc/passwd
DESCRIPTION
The file /etc/passwd is a local source of information about users'
accounts. The password file can be used in conjunction with other
password sources, such as the NIS maps passwd.byname and passwd.bygid and
the NIS+ table passwd.
.....................................................
```

Eléments de syntaxe ...

- Casse : MAJUSCULE ≠ minuscule
- Séparateur de mot : <espace>
- Caractères génériques (jokers)

$ cmde [-/+options] [argument(s)]

```
$ ls
$ ls -l
$ ls /dev
$ ls -l /dev
```

Il y a des exceptions ...

… éléments de syntaxe

Incorrecte	Correcte

1. Séparation :
$ mail - f newmail
$ who-u
	$ mail -f newmail
	$ who -u

2. Ordre :
$ mail newmail -f
$ stage01 mail
$ -u who
	$ mail -f newmail
	$ mail stage01
	$ who -u

3. Options multiples :
$ who -m-u
$ who -m u
	$ who -m -u
	$ who -mu

4. Arguments multiples $ mail stage01stage02	$ mail stage01 stage02

… session utilisateur

Ouverture de session :

```
login: stage01
stage01 Password: (le mot de passe n'apparaît pas)
$
```

Fin de session :

```
$ <Ctrl-d> (ou bien)
$ exit     (ou bien)
$ logout
login:
```

Utilisation du clavier

- Se limiter à la partie principale gauche:
  - pas de flèche,
  - pas de pavé numérique.
- Quelques actions clavier:

<ctrl-c>	Fin du process courant
<ctrl-\>	Fin du process courant + core
<ctrl-z>	Suspendre le process courant
<ctrl-u>	Annuler la ligne saisie
<crtl-d>	Terminer la saisie clavier
<entrée>	Fin de saisie et envoie de la ligne de commande
<ctrl-s>	Stopper le défilement
<ctrl-q>	Poursuivre le défilement

Changer son mot de passe

```
$ passwd
Changing password for "stage01"
stage01's Old password:
stage01's New password:
Enter the new password again:
$
```


1.7.f

Date et calendrier

```
$ date
Wed Mar 16 10:15:00 GMT 2011
$
```

```
$ cal 1 2003
                           January 2003
Sun    Mon   Tue      Wed   Thu    Fri    Sat
                       1     2      3      4
  5      6     7       8     9     10     11
 12     13    14      15    16     17     18
 19     20    21      22    23     24     25
 26     27    28      29    30     31
```

Qui a ouvert une session et identité ...

```
$ who
      root                 lft0          Sept 4 14:29
      team01               pts/0         Sept 4 17:21
```

```
$ who am i
team01           pts/0  Sept 4 17:21
$ whoami
team01
```

… identité et terminal

Liste résumée et nombre d'utilisateurs :

```
$ who -q
mike root mike
# users=3
$
```

Identité courante :

```
$ id
uid=125(stage01)gid=200(stage)
$
```

Nom de la machine sur le réseau :

```
$ hostname
solaris9
$
```

Nom du terminal :

```
$ tty
/dev/pts/2
$
```

1.7.*

Affichage de texte

```
$ echo Bienvenue chez Global Knowledge
Bienvenue chez Global Knowledge
$
$ echo -e "\tPremiere ligne\n\tDeuxieme ligne"
    Premiere ligne
    Deuxieme ligne
$
```

\n → saut de ligne supplémentaire
\b → retour arrière
\c → annulation du saut de ligne final
\t → tabulation
\r → retour chariot
\0n → caractère de code ascii n (en octal)

Éventuellement l'option –e pour que les caractères précédés par \ soient interprétés.

```
$ banner unix
#     #   #     #       #     #     #
#     #   ##    #       #       #   #
#     #   # #   #       #         ##
#     #   #  #  #       #         ##
#     #   #    ##       #       #   #
 ####    #     #       #     #     #
$
```

Pour pratiquer en TP – le courrier : mail 1/2

Ecrire :

```
$ mail team01
Subject:      Meeting
There will be a brief announcement meeting today
in room 602 at noon.
<Ctrl-d>
Cc:  <Enter>
$
```


TP1&2

Pour pratiquer en TP – le courrier : mail 2/2

Ouvrir sa b.à l., et gérer son courrier :

```
$ mail
Mail  [5.2 UCB] [AIX5.X]  Type ? for help
"/var/spool/mail/team01": 2 messages 1 new
 U 1 team05         Tue Jan 4  10:50   10/267  "Hello !"
>N 2 team02         Wed Jan 5  11:25   16/311  "Meeting"
? t 2
From team02  Wed Jan 5 11:25 2011
Date: Wed 5 Jan 2011 11:25
From: team02
To: team01
Subject: Meeting
Cc:
There will be a brief announcement meeting today in room 602 at noon.
? d
? q
```

Supprimer le message courant
Quitter sa b. à l.

TP1&2

Pour pratiquer en TP – messagerie instantanée : `write`

Stage01 écrit à stage02 :

```
$ whoami
stage01
$ write stage02
Bien le bonjour,
Ok pour ce midi ?
<Ctrl-d>
$
```

Stage02 reçoit :

```
$
Message de stage01 sur pts/2 à 16:16 ...
Bien le bonjour,
Ok pour ce midi ?
EOF
```

En complément – refuser / accepter de recevoir des messages par `write` *:*

Refuser :

```
$ mesg n
```

Accepter :

```
$ mesg y
```

TP1&2

Récapitulatif de quelques commandes

banner*	→	Affichage en gros caractères
cal	→	Affichage d'un calendrier
clear	→	Effacement de l'écran
date	→	Affichage de la date et de l'heure
echo	→	Affichage de texte
hostname	→	Affichage du nom réseau
id	→	Identification d'un utilisateur
passwd	→	Choix d'un mot de passe
tty	→	Affichage du nom du terminal
who	→	Liste des utilisateurs connectés

* `banner` *n'est pas toujours installé par défaut*

1.7.h

mise en pratique ...

TPs 1 & 2 :

Ouverture de session

et

premières commandes

1.8.*

Global Knowledge

2 – Systèmes de fichiers

Pour unix « tout est fichier »

volume physique → un disque découpé en partition(s)

volume logique → { volume(s) physique(s) }

système de fichiers → organisation d'un volume logique:

- table descriptive des fichiers
- blocs des fichiers
- [données de journalisation]

2.1

… « tout est fichier »

Fichiers pour l'accès aux matériels → `/dev`

Exemples :

- /dev/sda1 → 1er disque SATA ou SCSI, 1ere partition (Linux)
- /dev/dsk/c0d0s0 → accès au disque dur (AIX, Solaris, ...)
  - c0 → 1er contrôleur
  - d0 → 1er disque
  - s0 → 1ere partition
- /dev/cdrfs, /dev/iso9660 → cdrom
- /dev/tty01 → terminal utilisateur
- /dev/pts/1 → pseudo terminal (réseau, graphique)

2.1

Quelques Files Systems

Non journalisés :

- System V (n'est plus utilisés)
- UFS (Unix File System)
- ext2 (EXTended)

Journalisés :

- jfs (Journal File System)
- ext3, ext4, xfs

En réseau :

- NFS (Network File System)

Pseudo-FS :

- procfs, tmpfs …

2.2.*

Un fichier c'est :

- Un ensemble de données,

- Une suite de caractères ou d'octets divers ,

- Une adresse de début, une longueur,

- Aucune structure n'est imposée par l'OS.

2.2.*

Les différents type de fichier

- Fichiers ordinaires
  - texte, image, binaires exécutable, script, …
- Répertoires
  - contiennent les noms pour accéder aux fichiers.
- Spéciaux
  - associés au pilote (driver) des matériels.
- Liens symboliques
  - références vers FS différents, vers répertoire.

Noms de fichiers

- minuscules ≠ MAJUSCULES

- pas d'extension .xxx

- seul caractère « interdit » : le '/'

- le '.' au début du nom → fichier « caché »

- chaîne de caractère stockée dans répertoire contenant et est associé au n° inode du fichier ...

… inode d'un fichier

- identifiée par son numéro unique par FS,
- dans un répertoire son numéro est associé au nom,
- contient les caractéristique d'un fichier :
    - type,
    - droits d'accès,
    - nombre de liens physiques,
    - utilisateur et groupe propriétaire,
    - taille,
    - adresse des blocs qui constituent le fichier,
    - date de dernière modification,
    - date de dernier accès,
    - date de dernière modification de l'inode.

2.3.b

Fichiers répertoires

- table qui associe n° inode ↔ nom de fichier

- au moins deux entrées :

  '.' le répertoire courant lui même

  '..' le répertoire parent contenant le répertoire courant

- l'ancêtre commun est la racine d'arborescence → '/'

(rem : racine → root en anglais)

2.3

Fichier – chemin d'accès

Deux façons de nommer un fichier:

- référence absolue: commence par '/' et contient tous les répertoires intermédiaires :

  /home/stage01/monfichier

- référence relative: à partir du répertoire courant :

  ./monfichier

  monfichier

  . ./. ./usr/bin/date

2.3.b

Parcourir l'arborescence

Où suis – je ?

```
$ pwd
/home/stage01/perso
$
```

Me déplacer mode absolu

```
$ cd /usr
$ pwd
/usr
```

Me déplacer mode relatif

```
$ pwd
/usr
$ cd bin
$ pwd
/usr/bin
```

Retour à la maison

```
$ pwd
/usr/bin
$ cd
$ pwd
/home/stage01
```

Liste des fichiers

Syntaxe :

ls [-l d i t a R] [fichiers(s)]

- affiche la liste des fichiers cités.
- si nom de répertoire(s), alors leur(s) contenu(s).
- si pas de fichier cité, alors contenu du répertoire courant.

Exemple :

```
$ ls
c doc manuals test1

$ ls -a
. .. .profile c doc manuals test1
```

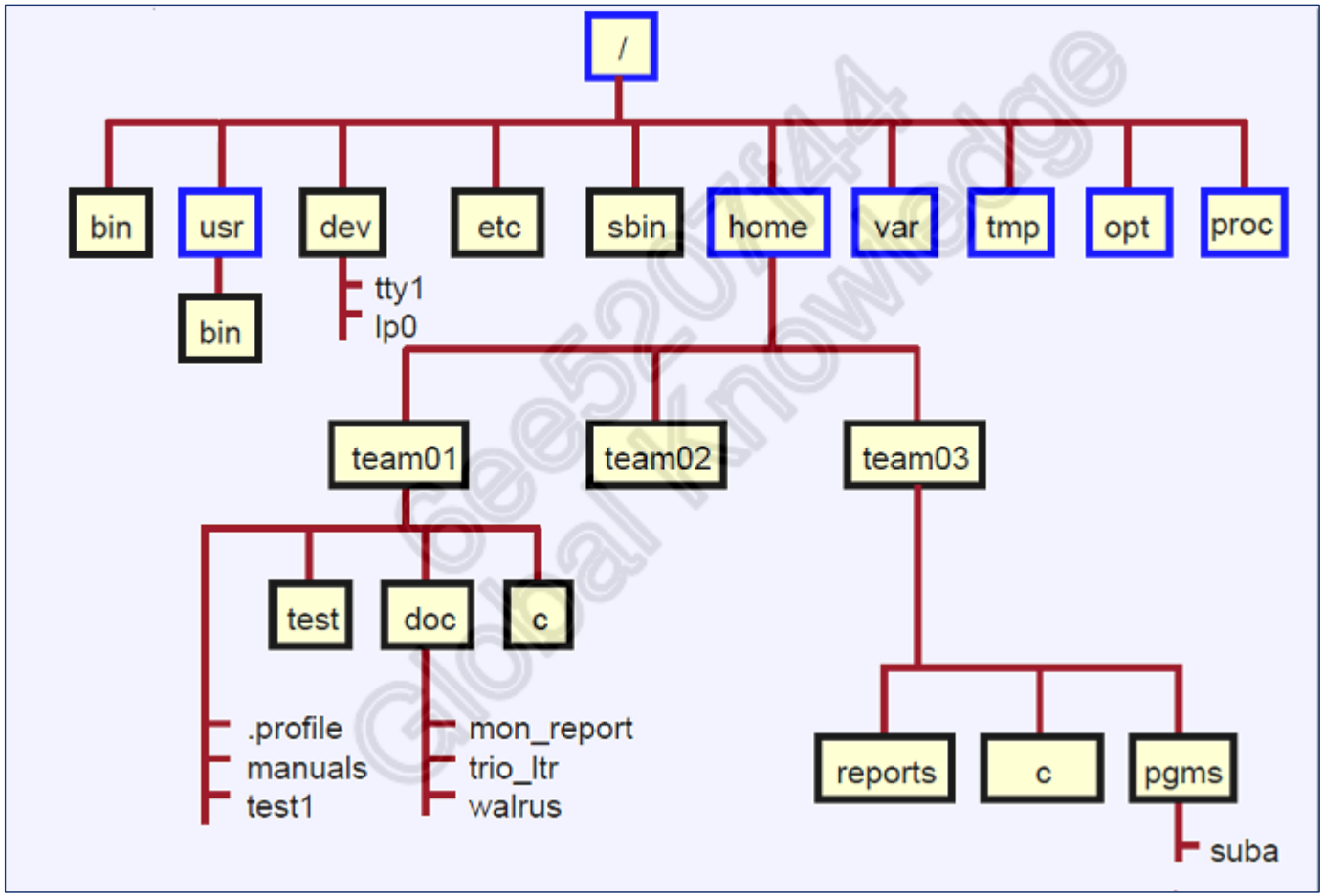
Liste des attributs : `ls -l`

```
$ ls -l /etc
total 504
.....................................................
lrwxrwxrwx  1 root  root   14 Jan 14  2004 aliases -> ./mail/aliases
drwxr-xr-x  2 root  bin   512 Jan 14  2004 apache
.....................................................
-r--------  1 root  sys   478 Sep 12 11:01 shadow
.....................................................
```

```
$ ls -l
total 0
$ ls -al
total 8
drwxr-xr-x   2  mike   mike   512   Sep 12 16:43 .
dr-xr-xr-x  12  root   root   512   Sep  1 20:55 ..
-rw-r--r--   1  mike   mike   144   Aug 25 16:26 .profile
-rw-------   1  mike   mike     8   Sep 12 16:44 .sh history
$
```

2.4 .*

Arborescence



2.4.b

Gestions des répertoires

mkdir → création d'un répertoire,

rmdir → suppression d'un répertoire à condition qu'il soit vide

```
$ mkdir rep
$ ls -ld rep
drwxr-xr-x   2   mike   mike   512   jan 23 12:26   rep
$
```

```
$ rmdir rep
rmdir: directory "rep": Directory not empty
$ cd rep/rep2
$ rmdir rep3
$ cd ..
$ rmdir rep2
$ cd ..
$ rmdir rep
$ ls -ld rep
ls: rep: No such file or directory
$
```

2.4.e

mise en pratique ...

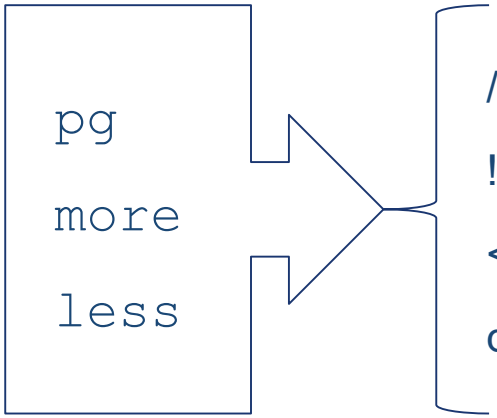
TP 3 – Nommage des fichiers et répertoires

2.5.*

2.6 Manipulations de fichiers ordinaires

Visualiser le contenu d'un fichier texte

`cat` → flux de texte sans pause d'affichage



/texte → recherche de texte en avant

!commande → lancement de commande unix

<rc> ou <espace> → avance d'une page

q → quitte

… autres options à découvrir via le man

Ces commandes sont très utiles pour contrôler l'affichage d'autres commandes.

2.6.a

Copie de fichiers : cp

Syntaxe :

cp [options] source cible
ou
cp [options] fic1 fic2 fic3 ... Repertoire

Exemple :

```
$ cp zorro zorrobis
$ ls -l zorro*
-rw-r--r--   1 mike   mike   1738 sep 12 12:23  zorro
-rw-r--r--   1 mike   mike   1738 sep 12 12:27  zorrobis
$
```

-i : demande de confirmation

```
$ ls -l toto
-rw-r-----   1 mike   mike    916 Sep 12 18:10  toto
$ cp -i toto zorro
cp: overwrite zorro (yes/no)? n
$
```


2.6.d

... cp

cible : un répertoire

```
$ ls -l toto
-rw-r-----   1 mike   mike    916 Sep 12 18:10  toto
$ mkdir rep
$ cp toto rep
$ ls -lR
.:
total 4
drwxr-x---   2 mike   mike    512 Sep 12 18:11  rep
-rw-r-----   1 mike   mike    916 Sep 12 18:10  toto
./rep:
total 2
-rw-r-----   1 mike   mike    916 Sep 12 18:11  toto
$
```

Liens : `ln`

Syntaxe :

ln [-s] source cible
ou
ln [-s] fic1 fic2 fic3 ... repertoire

a) Lien physique.
Intéret : différents chemins (noms de fichier) pour l'accès à une même entité physique.

Exemples :

```
$ ls -il toto
  150647 -rw-r--r--  1 mike  mike  30 Sep 20 12:22 toto
$ ln toto toto2
$ ls -il toto*
  150647 -rw-r--r--  2 mike  mike  30 Sep 20 12:22 toto
  150647 -rw-r--r--  2 mike  mike  30 Sep 20 12:22 toto2
$
```

2.6.j

Lien symbolique : `ln -s`

b) Liens symboliques.
Intérets :
- créer des liens entre FS ≠
- créer des liens vers des répertoires

Exemples :

```
$ ln -s toto stoto
$ ls -il *toto*
  150648 lrwxrwxrwx  1 mike  mike    4 Sep 20 12:23 stoto -> toto
  150647 -rw-r--r--  2 mike  mike   30 Sep 20 12:22 toto
  150647 -rw-r--r--  2 mike  mike   30 Sep 20 12:22 toto2
$
```

```
$ ln -s /home lhome
$ cd lhome
$ pwd
/home/mike/lhome
$ /usr/bin/pwd
/home
$
```

2.6.j

Renommer, déplacer des fichiers : `mv`

Syntaxe :

mv [options] source cible
ou
mv [options] fic1 fic2 fic3 ... repertoire

Exemple :

```
$ ls -il bernardo
  150634 -rw-r--r--  1 mike   mike   139 Sep 20 14:21 bernardo
$ mv bernardo zorro
$ ls -il zorro
  150634 -rw-r--r--  1 mike   mike   139 Sep 20 14:21  zorro
$
```


2.6.k

Renommer répertoire et lien symbolique

```
$ ls -il
  150636 lrwxrwxrwx 1 mike  mike  4   Sep 20 14:30 lzorro -> zorro
  150635 drwxr-xr-x 2 mike  mike  512 Sep 20 14:30 rep
  150634 -rw-r--r-- 1 mike  mike  139 Sep 20 14:30 zorro
$ mv rep repbis
$ mv lzorro szorro
$ ls -il
  150635 drwxr-xr-x 2 mike  mike  512 Sep 20 14:30 repbis
  150636 lrwxrwxrwx 1 mike  mike  4   Sep 20 14:30 szorro -> zorro
  150634 -rw-r--r-- 1 mike  mike  139 Sep 20 14:30 zorro
$
```

```
$ mv zorro toto
$ ls -il
  150635 drwxr-xr-x 2 mike  mike  512 Sep 20 14:30 repbis
  150636 lrwxrwxrwx 1 mike  mike  4   Sep 20 14:30 szorro -> zorro
  150634 -rw-r--r-- 1 mike  mike  139 Sep 20 14:30 toto
$ cat szorro
cat: cannot open szorro
$
```

Suppression de fichiers : `rm`

Syntaxe :

rm [options] fichier(s)

-i : demande de confirmation,
-f : force sans confirmation si il manque le droit d'écriture,
-r : suppression récursive de répertoire.

Exemple :

```
$ ls -il toto*
  150634 -rw-r--r--   2 mike   mike    139 Sep 20 14:57 toto
  150634 -rw-r--r--   2 mike   mike    139 Sep 20 14:57 toto2
$ rm toto2
$ ls -il toto*
  150634 -rw-r--r--   1 mike   mike    139 Sep 20 14:57 toto
$
```


2.6.I

Suppression de lien symbolique et de répertoire

Suppression de lien symbolique :

```
$ ls -il *toto
  150635 lrwxrwxrwx 1 mike  mike    4 Sep 20 15:04 stoto -> toto
  150634 -rw-r--r-- 1 mike  mike  139 Sep 20 14:57 toto
$ rm toto
$ ls -il *toto
  150635 lrwxrwxrwx 1 mike  mike    4 Sep 20 15:04 stoto -> toto
$ cat stoto
cat: cannot open stoto
$
```

-r : suppression récursive de répertoire

```
$ ls -l rep
-rw-r--r--   1 mike   mike   30 Sep 20 15:11 fic1
drwxr-xr-x   2 mike   mike  512 Sep 20 15:11 repbis
$ rm -r rep
$ ls -l rep
rep: No such file or directory
$
```

En tête de fichier : `head`

Exemple :

```
$ head -4 /etc/passwd
root:x:0:1:Super-User:/:/sbin/sh
daemon:x:1:1::/:
bin:x:2:2::/usr/bin:
sys:x:3:3::/:
$
```


2.6.n

Fin de fichier : `tail`

Exemple :

```
$ tail -4 /etc/passwd
stage5:x:5005:5000::/home/stage5:/usr/bin/ksh
stage6:x:5006:5000::/home/stage6:/usr/bin/ksh
stage7:x:5007:5000::/home/stage7:/usr/bin/ksh
stage8:x:5008:5000::/home/stage8:/usr/bin/ksh
$ tail -n +4 /etc/passwd
sys:x:3:3::/:
adm:x:4:4:Admin:/var/adm:
lp:x:71:8:Line Printer Admin:/usr/spool/lp:
uucp:x:5:5:uucp Admin:/usr/lib/uucp:
.............................
.............................
```


2.6.o

Comptage : wc

Exemples :

```
$ wc /etc/passwd
      23      35     916 /etc/passwd
$ wc -l /etc/passwd
      23 /etc/passwd
$ wc -w /etc/passwd
      35 /etc/passwd
$ wc -c /etc/passwd
     916 /etc/passwd
$
```

```
$ wc zorro*
       1       6      30 zorro
      41     469    1757 zorrobis
      42     475    1787 total
$
```


2.6.p

mise en pratique ...

TP 4 – Manipulations de fichiers

2.7.*

2.8 – Droits d’accès

Droits d'accès ...

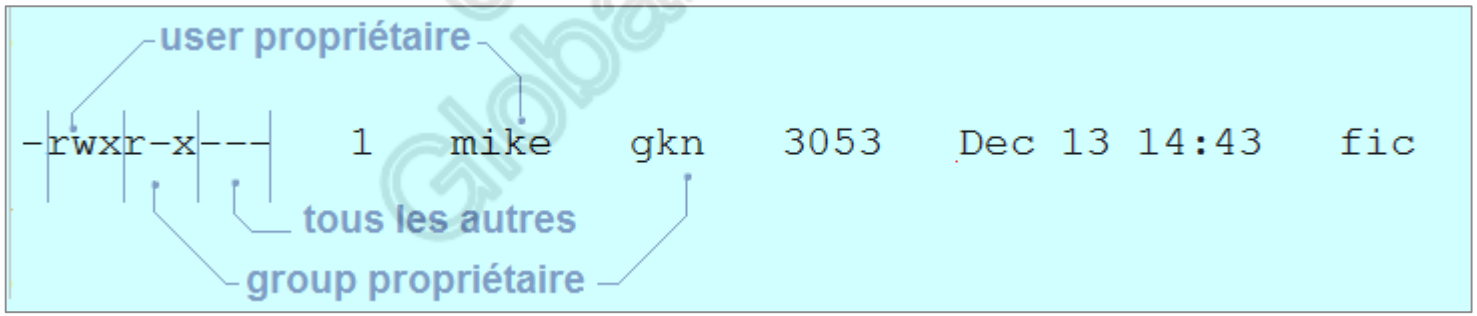
Tous les fichiers ont des droits d'accès.

trois attributs rwx :

- r : lecture,
- w : écriture ,
- x : exécute.

trois entités concernés :

- utilisateur propriétaire,
- groupe propriétaire,
- tous les autres.



2.8.a

… droits d'accès

Fichiers ordinaires :

- r → droit de lire le contenu
- w → droit de modifier le contenu
- x → droit d'exécution (si script alors besoin du droit r)

Répertoires :

- r → lister le contenu
- w → modifier le contenu (création et suppression d'entrée)
- x → droit de passage

Modifier les droits des fichiers existants : `chmod`

Syntaxes :

```
chmod   [ -R ]  mode_octal  fichiers ...
ou
chmod   [ -R ]  mode_symbolique  fichiers...
```

Mode octal :

r	→	$2^2 = 4$
w	→	$2^1 = 2$
x	→	$2^0 = 1$

Mode symbolique :

u	|		|	r
g	|	+	|	w
o	|	-	|	x

2.8.b

chmod – exemples 1

```
$ chmod 666 f1
$ ls -l f1
-rw-rw-rw-   1   mike   mike   0     Sep 20 18:01   f1
$ chmod go-w f1
$ ls -l f1
-rw-r--r--   1   mike   mike   0     Sep 20 18:01   f1
$ chmod g+w,o-r f1
$ ls -l f1
-rw-rw----   1   mike   mike   0     Sep 20 18:01   f1
$
```

chmod – exemples 2

supprimer un fichier → droit w sur le répertoire qui le contient

```
$ chmod 400 f1
$ ls -l f1
-r--------   1   mike   mike    0   Sep 20 18:01   f1
$ ls -ld
drwxr-xr-x   5   mike   mike   512   Sep 20 18:01   .
$ rm f1
rm: f1: override protection 400 (yes/no)? y
$ ls -l f1
f1: No such file or directory
$
```

Changer de propriétaire(s) – `chown`, `chgrp`

Condition: être user propriétaire, et membre group cible

(rem: ces commandes peuvent être réservées à l'usage de *`root`*.)

- **Changement du groupe propriétaire:**

$ `chgrp` *nveauGrpe* fic(s)

- **Changement de l'utilisateur propriétaire:**

$ `chown` *nveauUser* fic(s)

Récapitulatif des commandes … part 1

cat	→	Affichage ou concaténation de fichiers
cd	→	Changement de répertoire courant
chmod	→	Gestion des permissions d'un fichier
cp	→	Copie de fichiers
head	→	Affichage des premières lignes
less	→	Affichage page par page
ln	→	Création de liens
ls	→	Affichage du contenu d'un répertoire
mkdir	→	Création de répertoires
more	→	Affichage page par page

Récapitulatif des commandes … part 2

mv	→	Changement de nom ou déplacement de fichiers
pg	→	Affichage page par page
pwd	→	Nom du répertoire courant
rm	→	Suppression de fichiers
rmdir	→	Suppression de répertoires vides
tail	→	Affichage des dernières lignes d'un fichier
touch	→	Modification de la date d'un fichier avec création éventuelle
wc	→	Nombre de lignes, de mots et de caractères

mise en pratique ...

TP5 – Droits d'accès

2.10.*

Global Knowledge

Editeur `vi`

- Editeur commun à toutes les distributions Unix

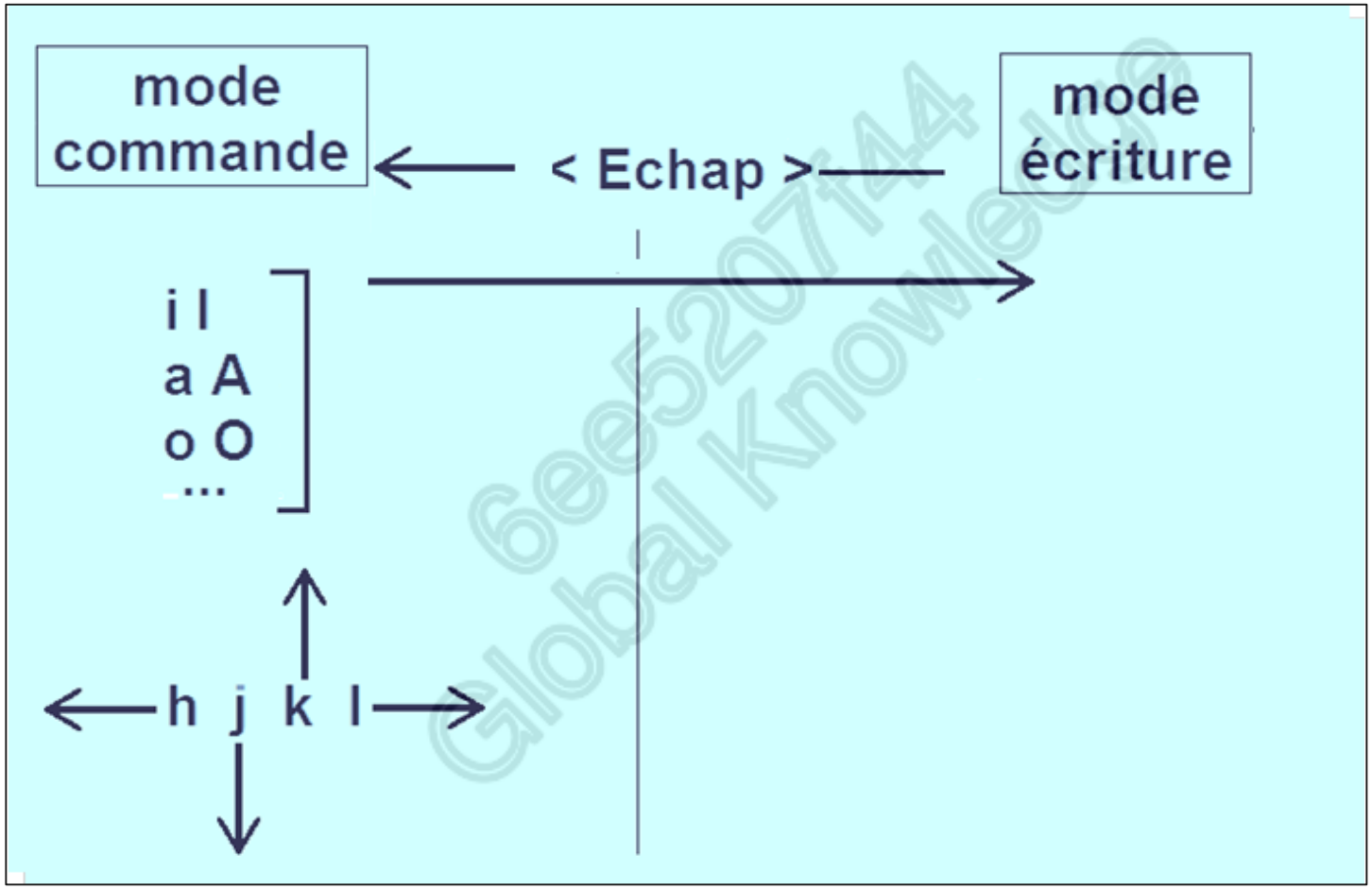
- Pour la création, la modification de fichiers de texte

- 3 modes de fonctionnements :

  - mode commande (mode actif au lancement),
  - mode écriture (dit aussi mode insertion)
  - mode commande syntaxique (mode ligne de commande)

3.1

vi – pratique globale



3.1

`vi` – principes généraux

- la plupart des commandes peuvent être multipliées

exemple: r → remplace 1 caractère

3r → remplace 3 caractères

- la commande en majuscule a une portée plus grande

exemple: a → écriture après la position courante du curseur

A → écriture à la fin de la ligne courante

- la plupart des commandes doublées concernent toute la ligne courante

exemple: dw → détruit le mot courant

dd → détruit la ligne courante

- toute destruction (suppression) est stockée dans un buffer (couper) qui pourra être inséré ensuite (coller)

`vi` – déplacement du curseur – part 1

h ou *n*h	→	Se déplacer vers la gauche (une ou *n* positions)
j ou *n*j	→	Se déplacer vers le bas (une ou *n* positions)
k ou *n*k	→	Se déplacer vers le haut (une ou *n* positions)
l ou *n*l	→	Se déplacer vers la droite (une ou *n* positions)
0 (zéro)	→	Se positionner en début de ligne
^	→	Se positionner sur le premier caractère significatif de la ligne
$	→	Se positionner en fin de ligne
b(egin)	→	Se positionner sur le mot précédent
w(ord)	→	Se positionner sur le mot suivant
*n*G	→	Aller à la ligne *n*
1G	→	Aller en début de fichier
G	→	Aller en fin de fichier

3.2.b

`vi` – déplacement du curseur – part 2

Ctrl d(own)	→	Descendre d'une moitié d'écran
Ctrl u(p)	→	Remonter d'une moitié d'écran
Ctrl f(orward)	→	Descendre d'un écran
Ctrl b(ack)	→	Remonter d'un écran
H(ome)	→	Aller en haut de l'écran
M(iddle)	→	Aller au milieu de l'écran
L(ast)	→	Aller en bas de l'écran
{	→	Aller au paragraphe précédent
}	→	Aller au paragraphe suivant
%	→	Trouver la prochaine parenthèse correspondante

3.2.a

`vi` – insertions, modifications, suppressions part 1

a	→	Insérer après le caractère courant (terminer par *Escape*)
A	→	Insérer en fin de ligne (terminer par *Escape*)
i	→	Insérer avant le caractère courant (terminer par *Escape*)
I	→	Insérer en début de ligne (terminer par *Escape*)
o	→	Insérer après la ligne courante (terminer par *Escape*)
O	→	Insérer avant la ligne courante (terminer par *Escape*)
x	→	Supprimer le caractère courant
s	→	Supprimer le caractère courant et passer en mode insertion
X	→	Supprimer le caractère précédent

3.2.b

`vi` – insertions, modifications, suppressions part 2

D	→	Supprimer jusqu'à la fin de la ligne
dd ou *n*dd	→	Supprimer une ou *n* lignes
dw ou *n*dw	→	Supprimer un ou *n* mots
dG	→	Supprimer jusqu'à la fin de fichier
r	→	Remplacer le caractère courant par un autre caractère
cw	→	Remplacer le mot courant (terminer par *Escape*)
C	→	Remplacer la fin de ligne (terminer par *Escape*)
R	→	Se placer en mode *sur-impression* (terminer par *Escape*)
J	→	Regrouper la ligne courante avec la ligne suivante

`vi` – recherche et substitution

/expr	→	Rechercher une expression vers le bas
?expr	→	Rechercher une expression vers le haut
n(ext)	→	Chercher l'occurrence suivante de l'expression
N(ext)	→	Chercher l'occurrence précédente de l'expression
:s/expr1/expr2/	→	Sur la ligne courante, substituer, une seule fois, la première expression par la seconde
:s/expr1/expr2/g	→	Sur la ligne courante, substituer, plusieurs fois si nécessaire, la première expression par la seconde
:n,ms/expr1/expr2/	→	Effectuer les remplacements entre les lignes *n* à *m*
:n,ms/expr1/expr2/g		

3.2.c

`vi` – copier / coller et autres commandes utiles

Y ou yy	→	Mémoriser la ligne courante dans un tampon de travail en vue d'un copier-coller
*n*Y ou *n*yy	→	Mémoriser *n* lignes à partir de la ligne courante
p (minuscule)	→	Insérer le tampon de travail après la position courante
P (majuscule)	→	Insérer le tampon de travail avant la position courante

3.2.e

u(ndo)	→	Annuler la dernière commande
.	→	Répéter la dernière commande
:!commande	→	Exécuter une commande Unix et revenir dans l'éditeur
!!commande	→	Insérer le résultat d'une commande (la ligne courante est écrasée)
Ctrl g	→	Obtenir le numéro de la ligne courante
Ctrl l ou Ctrl r	→	Rafraîchir l'écran en cas de parasites d'affichage

3.2.f

`vi` – sauve et quitte, et insertion fichier

:q(uitte) ou :q!	→	Quitter l'éditeur sans sauvegarder le fichier courant (Le *!* est nécessaire si des modifications ont eu lieu)
ZZ ou :wq ou :x	→	Sauver le fichier courant et quitter l'éditeur
:x!	→	Forcer la sauvegarde (fichier en lecture seule)
:w(rite)	→	Sauvegarder sans sortir
:w *fic*	→	Sauvegarder sous le nom *fic*
:*n1*,*n2*w fic	→	Sauvegarder les lignes de *n1* à *n2* dans le fichier de nom *fic*
:r(ead) fic	→	Insérer, après la ligne courante, le contenu du fichier *fic*

3.2.g

vi - paramétrage

Détails des paramètres via :

```
$ man ex
```

:set all	→	Afficher la liste des paramètres courant et leur valeur
:set option	→	Activer une option (exemple *:set autoindent*)
:set nooption	→	Inhiber une option (exemple *:set nonumber*)
:set opt=val	→	Donner une valeur de paramétrage (exemple *:set tabstop=4*)

Exemple de paramétrage permanent via `$HOME/.exrc` :

```
$ cat .exrc
set number
set autoindent
set tabstop=4
set showmode
$
```

3.2.i

mise en pratique ...

TP6 – Editeur vi

2.10.*

Global Knowledge

4 – Processus et mécanismes de redirections

Définitions

- un programme :

  → fichier ordinaire muni du droit x

- un process (ou tâche) :

  → un objet système

  → un programme en cours d'exécution + son environnement

  → peut comprendre plusieurs threads

- l'accès aux ressources est géré par le noyau qui octroie un temps partagé aux process.

- l'espace mémoire d'un process peut être copié dans un fichier nommé `core` (mécanisme de dump).

4.1

Attributs d'un process

- PID → Process IDentifier, numéro d'un process.
- PPID → Parent Process IDentifier, numéro du process parent.
- UID → User IDentifier et ...
- GID → ...Group IDentifier utilisés pour les droits d'accès.
- TTY → TeleTYpe terminal, identifie l'écran/clavier associé.
- NI → NIce value – intervient sur le calcul de la priorité initiale.
- STIME → Start TIME - heure de lancement.
- TIME → Temps CPU cumulé.

Visualiser les attributs de process : `ps` – part 1

Attributs simples des process associés au terminal courant :

```
$ ps
   PID TTY      TIME CMD
   463 pts/2    0:00 ksh
   479 pts/2    0:00 ps
$
```

Attributs détaillés des process associés au terminal courant :

```
$ ps -f
   UID   PID  PPID  C    STIME TTY      TIME CMD
  mike   463   461  0 13:50:50 pts/2    0:00 -ksh
  mike   480   463  0 15:09:43 pts/2    0:00 ps -f
$
```

Visualiser les attributs de process : `ps` – part 2

process de l'utilisateur *mike* et leurs attributs :

```
$ ps -fu mike
   UID   PID  PPID  C    STIME TTY          TIME CMD
  mike   331   309  0 09:31:54 ?           0:03 /usr/openwin/bin/Xsun :0
  mike   463   461  0 13:50:50 pts/2       0:00 -ksh
  mike   620   463  0 15:24:44 pts/2       0:00 ps -fu mike
  mike   547   533  0 15:20:39 pts/4       0:00 /usr/dt/bin/dtsession
  mike   551   547  0 15:21:04 ?           0:01 /usr/dt/bin/dtfile
  mike   549   547  0 15:20:40 ?           0:02 dtwm
  mike   611   610  0 15:24:17 ?           0:00 /usr/dt/bin/dtterm
  mike   613   611  0 15:24:17 pts/5       0:00 /usr/bin/ksh
```

Visualiser les attributs de process : `ps` – part 3

Tous les process et leurs attributs :

```
$ ps -ef
     UID   PID  PPID  C    STIME TTY      TIME CMD
    root     0     0  0 09:30:52 ?       0:03 sched
    root     1     0  0 09:30:53 ?       0:00 /etc/init -
    root     2     0  0 09:30:53 ?       0:00 pageout
    root     3     0  0 09:30:53 ?       0:00 fsflush
    root   338     1  0 09:31:58 ?       0:00 /usr/lib/saf/sac
    root   341   338  0 09:31:58 ?       0:00 /usr/lib/saf/ttymon
.....................................................
```

Signaux, interruption des process

$ **kill** [-n°signal] PID1 PID2 ...

N° signal :

- SIGINT (2) → <Ctrl>C
- SIGQUIT (3) → <Ctrl>\, idem signal 2 + création fichier « core »
- SIGTERM (15) → valeur par défaut. Arrêt normal « Soft »
- SIGKILL (9) → arrêt « Hard », NON détournable
- SIGHUP (1) → émit par le Shell qui se termine

4.5.a

Redirections

le terminal courant ...

```
$ tty
/dev/pts/2
$
```

… comprend 3 canaux de communications :

- entrée standard → canal 0 → La commande peut y lire des données.
- sortie standard → canal 1 → La commande peut y écrire des résultats.
- erreur standard → canal 2 → La commande peut y écrire des messages d'erreur.

remarque : pour savoir si une commande qui n'affiche rien s'est déroulée sans erreur, on testera son code retour $? (cf le chapitre base de la programmation shell)

Entrée / sortie standard

Exemple :
-la commande cat est faite pour afficher (écrire sur la sortie standard) le contenu de fichier(s) énumérés en argument d 'entrée.

- si aucun fichier n'est fourni en entrée, alors c'est l'entrée standard (le clavier) qui est lu en entrée.

```
$ cat
je n'ai pas fourni d'argument a la commande cat
je n'ai pas fourni d'argument a la commande cat
elle a donc decide de lire son entree standard
elle a donc decide de lire son entree standard
elle affiche en echo ses donnees sur la sortie standard
elle affiche en echo ses donnees sur la sortie standard
elle va se terminer sur un ctrl d en debut de ligne
elle va se terminer sur un ctrl d en debut de ligne
$
```

4.6

Redirections depuis / vers fichier ordinaire

entrée :

$ cmde **<** fic

sorties en création :

$ cmde **>** fic

$ cmde **2>** ficerr

sorties en ajout:

$ cmde **>>** fic

$ cmde **2>>** ficerr

Redirection d'entrée

```
$ cat message
J'ai prepare mon texte dans un fichier
afin de pouvoir utiliser une redirection de l'entree
lors de l'appel de la commande write
$ write mike < message
    Message from mike on solaris9 (pts/2) [ Wed Sep 22 10:21:23 ]
J'ai prepare mon texte dans un fichier
afin de pouvoir utiliser une redirection de l'entree
lors de l'appel de la commande write
<EOT>
$
```


4.6.a

Redirection de sortie

```
$ cat > fichier
je tape des lignes au clavier et je les stocke en meme temps dans un
fichier
je termine la saisie par un ctrl d en debut de ligne

$ cat fichier
je tape des lignes au clavier et je les stocke en meme temps dans un
fichier
je termine la saisie par un ctrl d en debut de ligne
$
```


4.6.b

Redirection sortie en ajout

```
$ cal 2 1959 > monfichier
$ echo "je rajoute du texte en fin de fichier" >> monfichier
$
$ cat monfichier
   February 1959
 S  M Tu  W Th  F  S
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
je rajoute du texte en fin de fichier
$
```


4.6.d

Redirections divers

A « NE PAS » faire :

```
$ > monfichier cal
$ cat monfichier
   September 2004
 S  M Tu  W Th  F  S
           1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
$
```

4.6.d

Élimination de la sortie – fichier « trou noir » /dev/null

```
$ echo "Bienvenue au club Unix"
Bienvenue au club Unix
$ echo "Bienvenue au club Unix" > /dev/null
$
```

4.6.e

Création de fichier par redirection

```
$ ls -l toto
toto: No such file or directory
$ > toto
$ ls -l toto
-rw-r--r--   1 mike   mike         0 Sep 22 15:07 toto

$ cal > toto
$ ls -l toto
-rw-r--r--   1 mike   mike       139 Sep 22 15:07 toto

$ > toto
$ ls -l toto
-rw-r--r--   1 mike   mike         0 Sep 22 15:07 toto
$
```


4.6.f

Protection contre l'écrasement accidentel

```
$ set -o noclobber
$ date > monfichier
ksh: monfichier: file already exists
$ date >| monfichier
$ set +o noclobber
$ cal > monfichier
$
```


4.6.g

Redirection de l'erreur standard

```
$ ls f1 f2
f2: No such file or directory
f1

$ ls f1 f2 > resul 2> erreur
$ cat resul
f1

$ cat erreur
f2: No such file or directory
$
```


4.6.h

Redirection simultanée sortie et erreur

```
$ ls f1 f2
f2: No such file or directory
f1

$ ls f1 f2 > resul 2>&1

$ cat resul
f2: No such file or directory
f1
$
```

« danger » :
pas d'espace avant le & ...

4.6.h

Processus séquentiels

```
$ id –a ; cal 2 1959
uid=2000(mike) gid=2000(mike) groups=2000(mike),3000(gkn)
   February 1959
 S  M Tu  W Th  F  S
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
$
```

Redirection processus séquentiels

```
$ ( id -a ; cal 2 1959 ; tty ) > toto
$ cat toto
uid=2000(mike) gid=2000(mike) groups=2000(mike),3000(gkn)
   February 1959
 S  M Tu  W Th  F  S
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
/dev/pts/2
$
```

Mécanismes du pipe ...

```
$ who
mike     pts/2        sept 22 15:17   (192.168.0.1)
root     pts/3        sept 22 17:42   (192.168.0.4)
mike     pts/4        sept 22 17:43   (localhost)
$ who | wc -l
       3
$
```

```
$ who
mike     pts/2        sept 22 15:17   (192.168.0.1)
root     pts/3        sept 22 17:42   (192.168.0.4)
mike     pts/4        sept 22 17:43   (localhost)
$ who | grep mike
mike     pts/2        sept 22 15:17   (192.168.0.1)
mike     pts/4        sept 22 17:43   (localhost)
$ who | grep mike | wc -l
       2
$
```

Mécanismes du pipe

```
$ ps -ef | head -5
     UID   PID  PPID  C    STIME TTY      TIME CMD
    root     0     0  0 07:50:52 ?       0:03 sched
    root     1     0  0 07:50:53 ?       0:00 /etc/init -
    root     2     0  0 07:50:53 ?       0:00 pageout
    root     3     0  0 07:50:53 ?       0:00 fsflush
$ ps -ef | tail -n +2 | head -5
    root     0     0  0 07:50:52 ?       0:03 sched
    root     1     0  0 07:50:53 ?       0:00 /etc/init -
    root     2     0  0 07:50:53 ?       0:00 pageout
    root     3     0  0 07:50:53 ?       0:00 fsflush
    root   338     1  0 07:51:58 ?       0:00 /usr/lib/saf/sac
$
```

Résultats intermédiaires : `tee` ...

```
$ who
mike     pts/2        sept 22 15:17   (monpc)
root     pts/3        sept 22 18:46   (solaris9)
mike     pts/5        sept 22 18:47   (localhost)
root     pts/4        sept 22 18:47   (solaris9)
$ who | grep mike | tee f1 | wc -l
       2
$ cat f1
mike     pts/2        sept 22 15:17   (monpc)
mike     pts/5        sept 22 18:47   (localhost)
$
```


4.8.a

Résultats intermédiaires : `tee`

```
$ ps -ef | tail -n +2 | head -5 | tee resul
    root     0     0  0 07:50:52 ?           0:03 sched
    root     1     0  0 07:50:53 ?           0:00 /etc/init -
    root     2     0  0 07:50:53 ?           0:00 pageout
    root     3     0  0 07:50:53 ?           0:00 fsflush
    root   338     1  0 07:51:58 ?           0:00 /usr/lib/saf/sac
$
```


4.8.a

Processus en arrière plan

```
$ ls -alR /usr  > resul  2> /dev/null  &
[1]     386
$ ps -f
   UID   PID  PPID  C    STIME TTY      TIME CMD
  mike   358   356  0 09:19:36 pts/2    0:00 -ksh
  mike   387   358  0 10:02:32 pts/2    0:00 ps -f
  mike   386   358 10 10:02:28 pts/2    0:02 ls -alR /usr
$
[1] +  Done(2)                 ls -alR /usr > resul 2> /dev/null &
$ ls -l resul
-rw-r--r--   1 mike   mike   5080786 Sep 23 10:03 resul
$
```

Arrière plan mode détaché: nohup

```
$ nohup  ls -alR /usr  > resul  2> /dev/null  &
[1]     428
$
$ exit
You have running jobs
$ exit
```

```
$ ps -fu mike
   UID   PID  PPID  C    STIME TTY      TIME CMD
  mike   428     1 28 10:15:55 ?        0:18 ls -alR /usr
  mike   436   431  0 10:16:44 pts/2    0:00 ps -fu mike
  mike   431   429  0 10:16:37 pts/2    0:00 -ksh
$
```


4.9.a

Contrôle des tâches – `jobs`, `fg` et `bg`.

jobs → Liste des tâches suspendues ou lancées en tâche de fond (&).

fg → Reprise de la dernière commande suspendue (par SIGTSTP, i.e. <Ctrl>Z)

fg %n → Reprise d'un processus désigné par son numéro dans la liste des tâches.

bg %n → Reprise, en arrière plan.

```
$ jobs
[1]- ls -lR / &
[2]+  Stoppé    sleep 20

$ fg %1
ls -lR /
lrwxrwxrwx.   1 root root    7 21 avril  2016 bin -> usr/bin
dr-xr-xr-x.   4 root root 4096 21 avril  2016 boot
...
```


4.9.b

Contrôle des tâches - exemples

```
$ ls -lR / > resul 2> /dev/null
^Z[1] + Stopped (SIGTSTP)          ls -lR / > resul 2> /dev/null
$ jobs
[1] + Stopped (SIGTSTP)          ls -lR / > resul 2> /dev/null
$ bg %1
[1]     ls -lR / > resul 2> /dev/null&
$ jobs
[1] +  Running                   ls -lR / > resul 2> /dev/null
$
```


4.9.b

Récapitulatif

grep → recherche d'expression

kill → envoie d'un signal à un process

nohup → protection contre le signal HUP (logout)

ps → liste des process

tee → dupplication sortie standard

write → messagerie instantanée

mise en pratique ...

TP7 – Gestion des process et redirections

4.11.*

Global Knowledge

5 – Utilisation du shell

Les différents shell - chronologie

- Bourne shell (sh ou bsh)
→ le plus ancien,
→ créé par Steve Bourne,
→ peu confortable pour un usage interactif (pas d'alias, pas de rappel de commandes…).

- C shell (csh)
→ issu des versions Unix Berkeley,
→ écrit par Bill Joy,
→ a apporté de nettes améliorations par rapport au Bourne shell (alias, rappel de commandes),
→ syntaxe des scripts semblable au langage C.

- Korn shell (ksh)
→ créé par David Korn.
→ compatibilité ascendante avec le Bourne shell,
→ fonctionnalités de programmation étendues (tableaux ...),
→ propose des améliorations qu'avaient apportées le C shell,
→ en production les scripts sont écrits aujourd'hui le plus souvent en Korn shell.

- Bourne again shell (bash)
→ logiciel libre,
→ shell par défaut des distributions Linux,
→ syntaxe très proche de celle du Korn shell.

5.1

Les variables ...

Leur nom :

- alphanumérique,
- celui des variables « système » sont en majuscule,
- on accède à leur contenu en le préfixant du "$".
- NE commence PAS par un chiffre,
- NE commence PAS par $.

Déclaration, initialisation, valorisation :

- le signe "=", sans espace.

```
$ var=toto
$ echo $var
toto
$
```


5.2.a

… Les variables

Si ambiguïté, alors ${*nomVar*} au lieu de *$nomVar* :

```
$ a=pa
$ b=ul
$ echo $a$b
paul
$ echo $a$bette
pa
$ echo $a${b}ette
paulette
$
```


5.2.a

Les variables : `set`, `env` et `export`

```
$ var1=toto
$ export var1
$ export var2=zorro
$ env | grep var
var1=toto
var2=zorro
$ var2=autrevaleur
$ env | grep var2
var2=autrevaleur
$
```


Modification de l'environnement du shell courant:

`$ . ./monscript`

(sera détaillé plus loin)

5.2.b

Variables "système"

HOME	→	Nom complet du répertoire de connexion
LOGNAME	→	Nom de connexion de l'utilisateur
OLDPWD	→	Nom du répertoire courant précédent, "cd -" pour y retourner
PATH	→	Liste des répertoires pour la recherche des commandes par le shell
PS1	→	Prompt principal (par défaut : $)
PS2	→	Prompt secondaire (par défaut : >)
PWD	→	Nom du répertoire courant
TERM	→	Type du terminal (émulation) utilisé notamment par l'éditeur vi
TMOUT	→	Nombre de secondes d'inactivité du shell avant déconnexion

5.2.c

Variables d'invite `PS1` et `PS2`

```
$ PS1='unix> '
unix> echo "Bienvenue au
> club Unix"
Bienvenue au
club Unix
unix> PS2='suite: '
unix> echo "Bienvenue au
suite: club Unix"
Bienvenue au
club Unix
unix>
```


5.2.c

Variable TERM

```
$ echo $TERM
xterm
$ unset TERM
$ clear
TERM environment variable not set.
$ TERM=bidon
$ clear
TERM environment variable not set.
$ export TERM
$ clear
'bidon': unknown terminal type.
$ TERM=xterm
$ clear
Effacement correct de l'écran
```


5.2.c

Variable PATH

```
$ PATH=/usr/local/bin
$ id
ksh: id:  not found
$ PATH=/usr/bin:$PATH
$ echo $PATH
/usr/bin:/usr/local/bin
$ id
uid=2000(mike) gid=2000(mike)
$
```

en complément : les commandes `whence` ou `whereis` :

```
$ whence date
/usr/bin/date
$ whence ifconfig
$ PATH=$PATH:/sbin
$ whence ifconfig
/sbin/ifconfig
$
```

5.2.c

Les caractères spéciaux

<	→	Redirection de l'entrée standard
> et **>>**	→	Redirections de la sortie standard (écrasement ou ajout)
2> et **2>>**	→	Redirections de l'erreur standard (écrasement ou ajout)
;	→	Processus séquentiels
|	→	Mécanisme du pipe
&	→	Mode arrière-plan
( )	→	Groupement de commandes
=	→	Affectation de variable
$	→	Contenu de variable
{ }	→	Contenu de variable et bloc de fonction

5.3.a

Noms fichiers : jokers-caractères génériques ...

***** → remplace n'importe quelle suite de caractères (même vide),

→ seul, il désigne tous les noms de fichiers du répertoire courant, sauf ceux qui commencent par '**.**' (point).

? → un caractère quelconque (non vide).

[ ] → un caractère parmi l'ensemble énuméré entre [ ],

le signe - (moins) pour un intervalle dans le code ascii (exemple: [a-z] désigne une lettre minuscule).

[!] → un caractère différent de l'ensemble énuméré entree [!] (la négation de [ ])

```
$ echo *
fic1 fic11 fic2 titi toto zorro
$ echo .*
. .. .exrc .profile .sh_history
$ echo .* *
. .. .exrc .profile .sh_history fic1 fic11 fic2 titi toto zorro
$
```

5.3.b

… Noms de fichier - jokers

```
$ echo .[!.]*
.exrc .profile .sh_history
$
```

```
$ echo *
fic1  fic11  fic2  toto  zorro  titi
$ echo fic*
fic1  fic11  fic2
$ echo fic?
fic1  fic2
$ echo *[12]
fic1  fic11  fic2
$ echo t[!o]*
titi
$ echo ?i*
fic1  fic11  fic2  titi
$ echo s*
s*
$
```


5.3.b

Substitution de commande

- But : dans une phrase de texte, remplacer une commande par son résultat.

- Deux syntaxes possibles :

  `commande`

  → la commande est encadrée par deux ` (quotes inverses).

  → syntaxe historique du Bourne Shell.

  $(commande**)**

  → depuis le Korn Shell : commande encadrée de parenthèses précédées du $.

  → fait mieux apparaître l'analogie avec la substitution de variable.

```
$ echo "Nombre de fichiers de $PWD : $(ls | wc -w)"
Nombre de fichiers de /home/mike :        6
$
```

5.3.c

Caractères de protection

\ → antislash : annule l'interprétation du seul caractère placé après lui.

' ' → apostrophe : tous les caractères situés entre deux simples quotes perdent leur rôle fonctionnel (excepté bien sur la quote elle-même).

" " → guillemets : tous les caractères situés entre deux doubles quotes perdent leur rôle fonctionnel, excepté la double quote et les trois caractères : $, `commande` et \.

```
$ echo *
fic1 fic11 fic2 titi toto zorro
$ echo \*
*
$ echo \\
\
$
```

```
$ echo <$LOGNAME>
ksh: syntax error: `newline or ;' unexpected
$ echo '<$LOGNAME>'
<$LOGNAME>
$ echo "<$LOGNAME>"
<mike>
$
```

```
$ echo "<\\$LOGNAME\$>"
<\mike$>
$
```

5.3.d

mise en pratique ...

TP8 – Eléments du Shell
variables et jokers

5.4.*

personnalisation de l’environnement

- Alias
- Historique des commandes
- Fichiers scripts de connexion

Alias : `alias` ...

Rem : apport du Korn Shell vis à vis du Bourne Shell.

```
$ alias l='/usr/bin/ls -l'
$ alias l
l='/usr/bin/ls -l'
$ l
total 6
-rw-r--r--   1 mike   mike         0 Sep 26 17:38 fic1
-r--r--r--   1 mike   mike       916 Sep 26 17:38 fic2
-rw-r--r--   1 mike   mike       106 Sep 26 17:38 toto
-r--r--r--   1 mike   mike       916 Sep 26 17:38 zorro
$
```

```
$ alias rm='/usr/bin/rm -i'
$ rm fic1
rm: remove fic1 (yes/no)? n
$ alias cp='/usr/bin/cp -i'
$ cp fic1 toto
cp: overwrite toto (yes/no)? n
$
```


5.5.a

Alias : `alias` et `unalias`

```
$ type type
type is a shell builtin
$ type cd
cd is a shell builtin
$ type l
l is an alias for /usr/bin/ls -l
$ type cal
cal is /usr/bin/cal
$
```

```
$ unalias l
$ l
ksh: l: not found
$
```

Historique des commandes

HISTFILE→	Nom du fichier d'historique	-	$HOME/.sh_history par défaut
HISTSIZE→	Nombre de commandes mémorisées	-	128 par défaut

5.5.b et 5.4.d

Historique des commandes : `alias fc` & r

Affichage des dernières commandes :

fc -l	→	Afficher les 16 dernières commandes avec un numéro d'ordre
history	→	Alias prédéfini pour *fc -l*, puis alias personnel : alias h=history
fc -l *n1 n2*	→	Afficher les commandes de numéro *n1* à *n2*

Relancer des commandes :

r	→	Relancer la dernière commande (*r* est un alias pour *fc -e -*)
r *num*	→	Relancer la commande d'indice *num* dans la liste
r *string*	→	Relancer la commande la plus récente commençant par *string*
r *st1=str2*	→	Relancer la dernière commande en remplaçant *str1* par *str2*

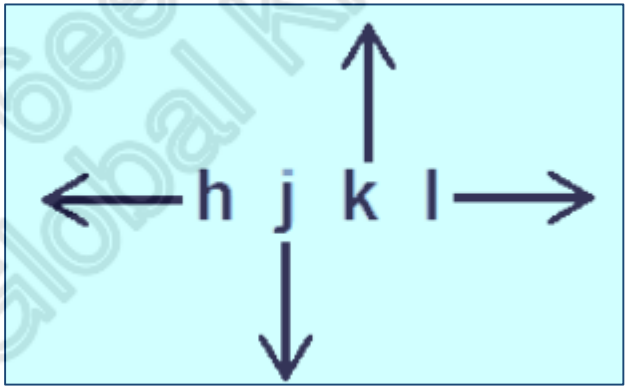
5.5.c

Historique des commandes – éditeur intégré

```
$ EDITOR=vi ; export EDITOR
```

ou bien :

```
$ set -o vi
```



5.5.d

Fichiers de connexion : $HOME/.profile

Exemple de fichier .profile :

```
# personnaliser les prompts
PS1="`hostname`.\$PWD\$ "
PS2="suite: "
# completer son PATH
PATH=$PATH:.
# se proteger contre les ecrasements via des redirections
set -o noclobber
# choix des droits par defaut
umask 027
# choix d'un second fichier (lu apres le .profile)
ENV=$HOME/.kshrc ; export ENV
```


5.6

Fichiers de connexion : ~/.kshrc

Exemple de fichier ~/.kshrc :

```
# mes alias
alias h=history
alias l='/usr/bin/ls -l'
alias rm='/usr/bin/rm -i'
alias cp='/usr/bin/cp -i'
alias mv='/usr/bin/mv -i'
alias p='/usr/bin/ps -fu $LOGNAME'
# historique des commandes
set -o vi
FCEDIT=$(whence vi) ; export FCEDIT
```


5.6

Fichiers de connexions – prise en compte des modifications

Pour éviter de se déconnecter / reconnecter

→ exécution dans le shell courant :

```
$ .   .profile
ksh: .profile:  not found
$ .   ./.profile
$
```


5.6.a

mise en pratique ...

TP9 – Personnaliser son environnement

5.7.*

Global Knowledge

Type d'un fichier : `file`

```
$ file toto titi rep
toto:           empty file
titi:           ascii text
rep:            directory

$ file /usr/sbin/*
/usr/sbin/6to4relay:    ELF 32-bit LSB executable 80386 Version 1,
dynamically linked, stripped
/usr/sbin/accept:       executable shell script
/usr/sbin/acctadm:      ELF 32-bit LSB executable 80386 Version 1,
dynamically linked, stripped
.....................................
```

```
$ file /bin
/bin:           directory

$ file -h /bin
/bin:           symbolic link to ./usr/bin
$
```


6.1.a

Numérotation des lignes : `nl`

```
$ nl /etc/passwd | tail
    14  mike:x:2000:2000::/home/mike:/usr/bin/ksh
    15  mysql:x:50:50::/home/mysql:/bin/sh
    16  stage1:x:5001:5000::/home/stage1:/usr/bin/ksh
    17  stage2:x:5002:5000::/home/stage2:/usr/bin/ksh
    18  stage3:x:5003:5000::/home/stage3:/usr/bin/ksh
    19  stage4:x:5004:5000::/home/stage4:/usr/bin/ksh
    20  stage5:x:5005:5000::/home/stage5:/usr/bin/ksh
    21  stage6:x:5006:5000::/home/stage6:/usr/bin/ksh
    22  stage7:x:5007:5000::/home/stage7:/usr/bin/ksh
    23  stage8:x:5008:5000::/home/stage8:/usr/bin/ksh
$
```


6.1.b

Comparer le contenu : cmp

Contenu identique → $? = 0

```
$ cp f1 f2
$ cmp f1 f2
$
```

Contenu différent → arrêt à la première différence

```
$ cat f1 >> f2
$ cmp f1 f2
cmp: EOF on f1
$
```


6.1.c

Diverses commandes complémentaires

comm	→	Comparaison de fichiers triés
diff	→	Comparaison de fichiers
fold	→	Découpage de lignes
join	→	Jointure entre deux fichiers triés
od	→	Visualisation de fichiers sous différents formats
paste	→	Concaténation de lignes entre plusieurs fichiers
pr	→	Mise en forme de fichiers pour impression
split	→	Éclatement d'un fichier en plusieurs fichiers

6.1.d

Recherche de fichiers : `find` ...

Syntaxe :

find repertoire(s) [option(s)] action(s)

Options :

-name fichier	→	Recherche sur le nom
-perm mode	→	Recherche sur les permissions
-newer fichier	→	Recherche des fichiers plus récents qu'un fichier donné
-user nomUser	→	Recherche sur le propriétaire
-mtime +n	→	Fichiers modifiés depuis plus de n jours
-mtime -n	→	Fichiers modifiés depuis moins de n jours
-atime +n	→	Fichiers consultés depuis plus de n jours
-atime -n	→	Fichiers consultés depuis moins de n jours
-inum num	→	Recherche sur un numéro d'inode
-type t	→	Recherche sur le type de fichier (t = b, c, d, f ou l)

6.2.a

... find

Actions :

-print	→	Affichage du nom des fichiers
-exec	→	Exécution d'une commande pour chaque fichier trouvé
-ok	→	Exécution d'une commande avec demande de confirmation

6.2.a

… find – exemples …

```
$ pwd
/home/mike
$ echo .??*
.Xauthority .profile .sh_history
$ find  /home  -name .??*  -print
find: bad option .Xauthority
find: path-list predicate-list
$ find  /home  -name '.??*'  -print
/home/mysql/.profile
/home/mike/.profile
/home/mike/.sh_history
/home/mike/.Xauthority
/home/stage1/.profile
/home/stage2/.profile
/home/stage3/.profile
$
```

```
$ find  /etc  /usr/bin  -name '?sh'  -print
find: cannot read dir /etc/inet/secret: Permission denied
find: cannot read dir /etc/sfw/private: Permission denied
/etc/ssh
/usr/bin/csh
/usr/bin/jsh
...
$ find  /etc  /usr/bin  -name '?sh'  -print  2> /dev/null
/etc/ssh
/usr/bin/csh
/usr/bin/jsh
...
$
```


6.2.a

… find – exemples

```
$ find  /home  -name .profile  -user mike  -print
/home/mike/.profile
$
```

```
$ find  /home  \( -name .profile  -o  -user mike  \)  -print
/home/mysql/.profile
/home/mike
/home/mike/.profile
/home/mike/.sh_history
/home/mike/.Xauthority
/home/mike/toto
/home/mike/titi
/home/mike/rep
/home/mike/rep/f1
...
$
```

```
$ find  .  -name '.??*'  -exec wc {} \;
       5       27      154 ./.profile
     169      644     3258 ./.sh_history
       0        1      101 ./.Xauthority

$ find  .  -name '.??*'  -ok wc {} \;
< wc ... ./.profile >?   y
       5       27      154 ./.profile
< wc ... ./.sh_history >?   n
< wc ... ./.Xauthority >?   y
       0        1      101 ./.Xauthority
$
```


6.2.a

Sauvegardes – commandes de compression

	Origine System V	Origine Berkeley	Logiciel libre
Commande de compression	pack	compress	gzip
Extension des fichiers compressés	.z	.Z	.gz
Commande de décompression	unpack	uncompress	gunzip
Décompression sur sortie standard	pcat	zcat	

Sauvegardes

archivage vers :

- → fichier spécial – device de sauvegarde
- → fichier ordinaire
- → sortie standard en vue de pipe
- → avec noms relatifs pour plus de souplesse

restauration :

- → avec les noms (path) archivés
- → nom relatifs pour liberté de choisir le répertoire de restauration

Sauvegardes : `tar`

Création d'archive :

tar -cvf *ficArchive* fichier(s)

Liste du contenu d'archive :

tar -tvf *ficArchive*

Restauration :

tar -xvf *ficArchive* [fichier(s)]

… tar - exemples

Sauvegarde :

```
$ tar cvf /tmp/montar .
a ./ 0K
a ./.profile 1K
a ./.sh_history 4K
a ./rep/ 0K
a ./rep/fic3 2K
...
$ file /tmp/montar
/tmp/montar:    USTAR tar archive
$
```

Listing :

```
$ tar  tvf  /tmp/montar
drwxr-xr-x 2000/2000      0 Sep 24 21:25 2004 ./
-rw-r--r-- 2000/2000    154 Sep 23 10:13 2004 ./.profile
-rw------- 2000/2000   4032 Sep 24 21:27 2004 ./.sh_history
drwxr-xr-x 2000/2000      0 Sep 24 21:25 2004 ./rep/
-rw-r--r-- 2000/2000   1242 Sep 24 21:24 2004 ./rep/fic3
...
$
```


6.3.c

... `tar` – exemple ...

Restauration complète:

```
$ pwd
/home/mike
$ rm -r *
$ ls
$ tar xvf /tmp/montar
x ., 0 bytes, 0 tape blocks
x ./.profile, 154 bytes, 1 tape blocks
x ./.sh_history, 4032 bytes, 8 tape blocks
x ./rep, 0 bytes, 0 tape blocks
x ./rep/fic3, 1242 bytes, 3 tape blocks
...
$ ls -alR
.:
total 24
drwxr-xr-x   3 mike   mike       512 Sep 24 21:25 .
dr-xr-xr-x  12 root   root       512 Sep  1 20:55 ..
-rw-r--r--   1 mike   mike       154 Sep 23 10:13 .profile
-rw-------   1 mike   mike      4032 Sep 24 21:27 .sh_history
drwxr-xr-x   2 mike   mike       512 Sep 24 21:25 rep
-rwxr-xr-x   2 mike   mike      1242 Sep 24 21:25 rep/fic3
................................................
```


6.3.c

... `tar` – exemple ...

Restauration partielle :

```
$ rm fic1
$ tar xvf /tmp/montar fic1
tar: 1 file(s) not extracted
$ tar xvf /tmp/montar ./fic1
x ./fic1, 1242 bytes, 3 tape blocks
$ ls -l fic1
-rw-r--r--   1 mike   mike      1242 Sep 24 10:19 fic1
$
```

… tar - exemple

tar + compression / décompression :

```
$ tar cvf - . | compress > /tmp/montar.Z
a ./ 0K
a ./.profile 1K
a ./.sh_history 5K
a ./rep/ 0K
a ./rep/fic3 2K
a ./fic2 1K
a ./.exrc 1K
a ./fic1 2K

$ uncompress < /tmp/montar.Z | tar xvf - ./fic1
x ./fic1, 1242 bytes, 3 tape blocks
$
```

Versions plus récentes : options `-z` ou `-j` → compression directement par `tar`

```
$ tar cvzf /tmp/montar.gz rep
```


6.3.c

Impressions

	Spooler System V	Spooler Berkeley	Spooler IBM/AIX
Requêtes d'impression	lp	lpr	qpr
Etat des files d'attente	lpstat	lpq	qchk
Etat des files d'attente	cancel	lprm	qcan

Trace d'une session : `script`

```
$ script
Script started, file is typescript
$ echo bonjour
bonjour
$ who -q
mike
# users=1
$ exit
Script done, file is typescript
$ cat typescript
Script started on Sat Sep 25 09:17:41 2004
$ echo bonjour
bonjour
$ who -q
mike
# users=1
$ exit

script done on Sat Sep 25 09:18:09 2004
$
```


6.5.a

Utilisation du disk : `du`

```
$ du
6       ./rep
34      .
$
$ du -ak
1       ./.profile
5       ./.sh_history
2       ./rep/fic3
3       ./rep
1       ./fic2
.........................................
$ du /home
10      /home/mysql
6       /home/mike/rep
34      /home/mike
.........................................
$ du -s /home
126     /home

$ du -ks /home
63      /home
$
```


6.5.c

Changer d'identité : `su`

```
$ id ; pwd
uid=2000(mike) gid=2000(mike)
/home/mike
$ su
Password:
# id ; pwd
uid=0(root) gid=1(other)
/home/mike
# echo $PATH
/usr/bin:/bin:.
# exit
$
```

```
$ su -
Password:
Sun Microsystems Inc.   SunOS 5.9       Generic January 2003
# id ; pwd
uid=0(root) gid=1(other)
/
# echo $PATH
/usr/sbin:/sbin:/usr/bin:/bin
# exit
$
```


6.5.d

Extraction de champs : `cut` ...

Rappel, format de `/etc/passwd` :

nomLogin : pwd : UID : GID : commentaire : répertoireAccueil : processConnexion

```
$ cut -d: -f 1,3,7 /etc/passwd | tail -5
stage4:5004:/usr/bin/ksh
stage5:5005:/usr/bin/ksh
stage6:5006:/usr/bin/ksh
stage7:5007:/usr/bin/ksh
stage8:5008:/usr/bin/ksh
$
```

```
$ echo Bonjour cher ami | cut -f 2,3
Bonjour cher ami
$ echo Bonjour cher ami | cut -d" " -f 2,3
cher ami
$
```


6.5.e

... cut

Plusieurs caractères séparateurs ...

```
$ ls -l fic1
-rw-r--r--   1 mike   mike      1242 Sep 24 10:19 fic1
$ ls -l fic1 | cut -d" " -f 3,5,9
mike

$ echo $(ls -l fic1)
-rw-r--r-- 1 mike mike 1242 Sep 24 10:19 fic1
$ echo $(ls -l fic1) | cut -d" " -f 3,5,9
mike 1242 fic1
$
```

Extraction mode caractères -c

```
$ echo Bonjour cher ami | cut -c 1-3,14-16
Bonami
$ echo Bonjour cher ami | cut -c 9-
cher ami
$
```

6.5.e

Tris : `sort` ...

Par défaut : tri le contenu par début de ligne, en mode ascii.

Exemple : tri numérique inversé entre les 2e et 3e délimiteurs de champs ":"

```
$ sort  -t:  +2nr -3  /etc/passwd  |  head
nobody4:x:65534:65534:SunOS 4.x Nobody:/:
noaccess:x:60002:60002:No Access User:/:
nobody:x:60001:60001:Nobody:/:
stage8:x:5008:5000::/home/stage8:/usr/bin/ksh
stage7:x:5007:5000::/home/stage7:/usr/bin/ksh
stage6:x:5006:5000::/home/stage6:/usr/bin/ksh
stage5:x:5005:5000::/home/stage5:/usr/bin/ksh
stage4:x:5004:5000::/home/stage4:/usr/bin/ksh
stage3:x:5003:5000::/home/stage3:/usr/bin/ksh
stage2:x:5002:5000::/home/stage2:/usr/bin/ksh
$
```


6.6.a

… tris

Après avoir éliminé la ligne de titre de ls -l,
tri principal numérique inverse sur la taille,
et tri secondaire sur le nom :

```
$ ls -l | tail -n +2 | sort +4nr -5 +8b -9
-rw-r--r--   1 mike   mike     1242 Sep 24 10:19 exemple
-r--r--r--   1 mike   mike      916 Sep 24 12:29 fic2
-r--r--r--   1 mike   mike      916 Sep 25 12:30 zorro
drwxr-xr-x   2 mike   mike      512 Sep 24 21:44 rep
-rw-r--r--   1 mike   mike      149 Sep 25 09:18 typescript
-rw-r--r--   1 mike   mike      106 Sep 25 09:23 fic
-rw-r--r--   1 mike   mike      106 Sep 25 09:23 ficbis
-rw-r--r--   1 mike   mike      106 Sep 25 12:30 toto
$
```

6.6.a

Transformation de caractères : `tr`

Transformation d'une plage de valeurs :

```
$ echo bienvenue au club Unix | tr '[a-z]' '[A-Z]'
BIENVENUE AU CLUB UNIX
$
```

Suppression d'un caractère :

```
$ echo bienvenue au club Unix | tr -d " "
bienvenueauclubUnix
$
```


6.6.b

Filtre – Sélection – de lignes : `grep`, `egrep` ...

Syntaxes :

grep [options] expression fichier(s)

egrep [options] expr1 | expr2 fichier(s)

Principales options :

-i → confond minuscules / Majuscules,

-v → négation : affiche les lignes qui NE contiennent PAS expression(s) ,

-c → (compte) affiche le nombre de lignes trouvées,

-l → affiche uniquement le nom de fichier,

-f → expressions écrites dans un fichier ...

6.6.c

…`grep` – Expressions régulières ...

^	→	Début de ligne
$	→	Fin de ligne (se place en fin d'expression)
.	→	Présence d'un caractère quelconque
[ ]	→	Un caractère parmi un ensemble
[^]	→	Un caractère ne figurant pas dans l'ensemble
*	→	Un nombre quelconque d'apparitions d'un caractère (y compris aucun)
^$	→	Ligne vide (un début et une fin tout de suite)
.*	→	Reste de la ligne ou ligne complète suivant le contexte
|	→	Le caractère | signifie OU et relie deux expressions (`egrep` seulement)

6.6.c

... grep, egrep - exemples

Les lignes qui commencent par une lettre majuscule :

```
grep '^[A-Z]' fichier
```

Les lignes qui se terminent par 'cd1' et un nombre quelconque de " " :

```
grep 'cd1 *$' fichier
```

Lignes qui commencent par une majuscule ou bien qui se terminent par cd1 :

```
egrep '^[A-Z]|cd1$' fichier
```

Lignes qui commencent par une majuscule et qui se terminent par la chaîne cd1 :

```
grep '^[A-Z].*cd1$' fichier
```

6.6.c

Récapitulatif des commandes … 1/3

cmp	→	Comparaison de deux fichiers
comm	→	Comparaison de fichiers triés
compress, uncompress	→	Compression, décompression de fichiers
cpio	→	Archivage et restauration de fichiers
cut	→	Sélection de caractères ou de champs
diff	→	Comparaison de fichiers
du	→	Informations sur l'espace disque utilisé
file	→	Détermination du type de contenu
find	→	Recherche de fichiers
fold	→	Découpage de lignes
grep, egrep	→	Recherche d'expressions

Récapitulatif des commandes … 2/3

gzip, gunzip	→	Compression, décompression de fichiers
join	→	Jointure entre deux fichiers triés
lp, lpstat, cancel	→	Commandes d'impression du spouleur System V
lpr, lpq, lprm	→	Commandes d'impression du spouleur Berkeley
nl	→	Numérotation de lignes
od	→	Visualisation de fichiers sous différents formats
pack, unpack	→	Compression, décompression de fichiers
paste	→	Concaténation de lignes entre plusieurs fichiers
pr	→	Mise en forme de fichiers pour impression
qprt, qchk, qcan	→	Commandes d'impression du spouleur AIX
script	→	Capture d'une session sur un terminal

Récapitulatif des commandes ... 3/3

sort	→	Utilitaire de tri
split	→	Éclatement d'un fichier en plusieurs fichiers
su	→	Changement d'identité
tar	→	Archivage et restauration de fichiers
tr	→	Transformation de caractères

mise en pratique ...

TP10 – Utilitaires

5.4.*

Global Knowledge

7 – Commandes réseau, environnements graphiques

Interfaces physiques : `ifconfig`

Exemple HP-UX :

```
$ ifconfig  lan0
lan0: flags=843<UP,BROADCAST,RUNNING,MULTICAST>
      inet 172.16.1.42  netmask ffff0000  broadcast 172.16.255.255
$
```

Exemple AIX:

```
$ ifconfig  en0
en0:flags=8080863<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST>
      inet 172.16.1.10  netmask 0xffff0000  broadcast 172.16.255.255
$
```

Exemple Linux :

```
$ ifconfig  -a
eth0    Link encap:Ethernet  HWaddr 00:60:97:FE:CB:96
        inet addr:172.16.1.2  Bcast:172.16.255.255  Mask:255.255.0.0
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

        ..............
lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        UP LOOPBACK RUNNING  MTU:16436  Metric:1
        ..............
```

7.1

Interfaces physiques : `ifconfig`, `netstat -i`

Exemples Solaris :

```
$ ifconfig  -a
lo0: <UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
    inet 127.0.0.1  netmask ff000000
elxl0: <UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
    inet 172.16.1.3  netmask ffff0000  broadcast 172.16.255.255
    ether 0:b0:d0:1d:9:c1
```

```
$ netstat  -in
Name  Mtu  Net/Dest   Address   Ipkts Ierrs Opkts Oerrs Collis Queue
lo0   8232 127.0.0.0  127.0.0.1   384     0   384     0      0     0
elxl0 1500 172.16.0.0 172.16.1.3 9567     0  4093     0      0     0
$
```

Résolution des noms – résolution locale

Fichier `/etc/hosts` :

```
# Internet host table
127.0.0.1       localhost
172.16.1.3      solaris9  loghost
172.16.1.240    aix43  ibm1     # serveur NFS
172.16.1.242    aix5  ibm2
172.16.1.245    hpux11           # machine de test
172.16.1.246    redhat9
#
```


7.2.a

Résolution des noms – Domain Name System

- Client DNS - Fichier `/etc/resolv.conf` :

```
domain          masociete.com
nameserver      172.16.1.1
nameserver      172.16.2.10
```

- Ordre de recherche entre `/etc/hosts` (files) et `/etc/resolv.conf` (dns) : `/etc/nsswitch.conf`

```
hosts:   files   dns
```

- Commandes d'interrogation interactive :
`nslookup, dig`

7.2.b

Terminal virtuel : `telnet`

```
$ telnet solaris9
Trying 192.168.0.6...
Connected to solaris9.
Escape character is '^]'.

SunOS 5.9
login: mike
Password:
Last login: Sat Sep 25 13:41:04 from 192.168.0.1
Sun Microsystems Inc.   SunOS 5.9       Generic January 2003
$
.....................................
$ exit
Connection to solaris9 closed by foreign host.
$
```


7.3.a

Les "remotes commands"

`rlogin` → connexion distante,

`rsh (rcmd)` → commandes distantes,

`rcp` → copie de fichiers

- Spécifiques aux environnements Unix,
- Présence d'un fichier `$HOME/.rhosts` sur machine distante,
- Eventuellement `/etc/hosts.equiv`.

Rem : Intérêt historique uniquement. Premières commandes réseau de Berkeley. Remplacées par `ssh` et `scp` pour des raisons de sécurité essentiellement .

7.3.b

Commande distante : `rsh` ou `rcmd`

```
$ cat toto | rsh solaris9 lp
request id is impr-989 (standard input)
$ rsh solaris9 lpstat
impr-989        mike                75   Sep 24 12:10
$
```

`$rsh solaris9 cal 2004 > toto`

→ fichier créé sur la machine locale

`$rsh solaris9 'cal 2004 > toto'`

→ fichier créé sur la machine distante

7.3.d

Copie de fichiers : `rcp`

```
$ rcp  toto  solaris9:/tmp
$ rsh  solaris9  'ls -l /tmp/toto'
-rw-r--r--   1 mike   mike   1264   Sep 14 18:45   toto
$
```


7.3.e

Transfert de fichier : `ftp` …

Syntaxe :

ftp [options] host

Exemple de démarrage d'une session `ftp` :

```
$ ftp  solaris9
Connected to solaris9.
220 solaris9 FTP server ready
Name (solaris9:mike):
331 Password required for root.
Password:
230 User mike logged in.
ftp>
```

Transfert de fichier : `ftp` – les commandes 1/2

Commandes liées à l'environnement :

	help ou ?	→	Aide en ligne
	status	→	Affichage de l'état courant de la session
	ascii	→	Transferts en mode texte (bonne conversion entre Unix et Windows)
→	binary	→	Transferts en mode binaire (recommandé entre deux systèmes Unix)
	pwd	→	Nom du répertoire courant sur la *machine distante*
→	cd rep	→	Changement de répertoire sur la *machine distante*
→	ls	→	Liste des noms de fichiers sur la *machine distante*
	!pwd	→	Affichage du nom du répertoire courant sur la *machine locale*
→	lcd rep	→	Changement de répertoire sur la *machine locale*
→	!ls	→	Liste des noms de fichiers sur la *machine locale*

7.3.f

Transfert de fichier : `ftp` – les commandes 2/2

Commandes de transfert de fichiers et de fin de session ftp :

get *f1 f2*	→	Récupération d'un fichier distant *f1* vers fichier local *f2*
recv *f1 f2*	→	Récupération d'un fichier (synonyme de get)
put *f1 f2*	→	Émission d'un fichier local *f1* vers un fichier distant *f2*
send *f1 f2*	→	Émission d'un fichier (synonyme de put)
glob	→	Réactive ou inhibe la résolution des caractères spéciaux
prompt	→	Activation du mode interactif pour un transfert multiple
mget *fichiers*	→	Récupération de fichiers vers le répertoire local courant
mput *fichiers*	→	Émission de fichiers vers le répertoire distant courant
bye	→	Sortie de ftp
quit	→	Sortie de ftp (synonyme de bye)

7.3.f

Session ftp - exemple

```
ftp> status
Connected to solaris9.
No proxy connection.
Mode: stream; Type: ascii; Form: non-print; Structure: file
Verbose: on; Bell: off; Prompting: on; Globbing: on
Store unique: off; Receive unique: off
Case: off; CR stripping: on
Ntrans: off
Nmap: off
Hash mark printing: off; Use of PORT cmds: on
ftp> binary
200 Type set to I.
ftp> get  /etc/group  /tmp/toto
local: /tmp/toto remote: /etc/group
200 PORT command successful.
150 Binary data connection for /etc/group (192.9.200.130,1069) (1264
bytes).
226 Binary Transfer complete.
1264 bytes received in 0.03 seconds (41 Kbytes/s)
ftp> mget  /home/mike/.??*hrc
mget /home/mike/.exrc? y
200 PORT command successful.
150 Binary data connection for /home/mike/.exrc (192.9.200.130,1073) (196
bytes).
226 Binary Transfer complete.
196 bytes received in 0 seconds (0.19 Kbytes/s)
mget /home/stage1/.kshrc? n
ftp> quit
221 Goodbye.
$
```


7.3.f

Alternative sécurisée `ssh`

SSH : Secure SHell

→ mot de passe chiffré

→ communication chiffrée

→ possibilité d'authentification par certificat

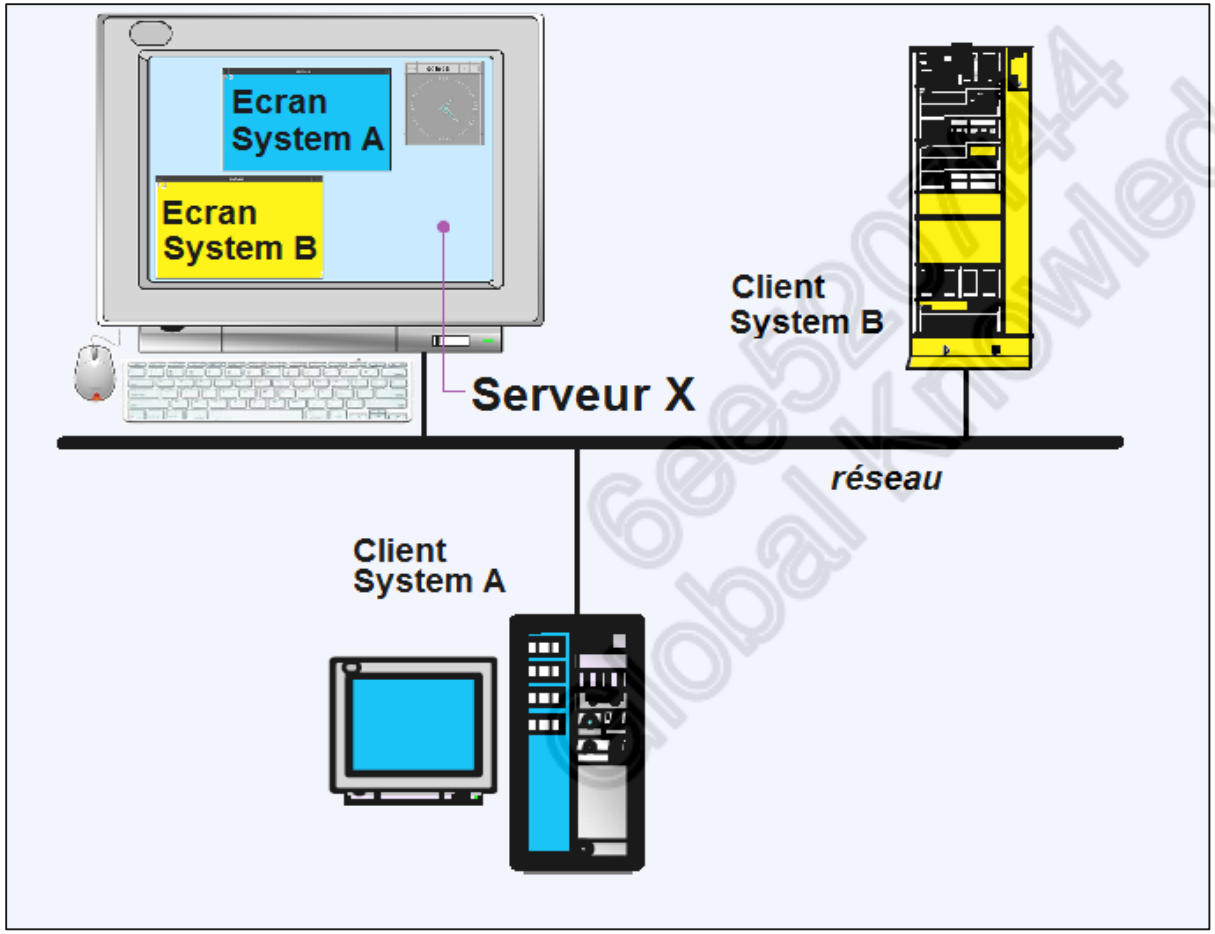
Commandes historiques	Commandes alternatives `ssh`

`telnet`
`rlogin`
`rsh`
	`ssh`

`rcp` `ftp`	`scp` `sftp`

7.3.g

Environnement graphique – X11 / X-Window



- Protocole XWindow / X11

- Serveur X :

→ un process accessible par le réseau,

→ une machine spécialisée : terminal X,

→ @IP et N° mémorisés dans la variable `DISPLAY`.

7.4.a

Environnement – Desktop …

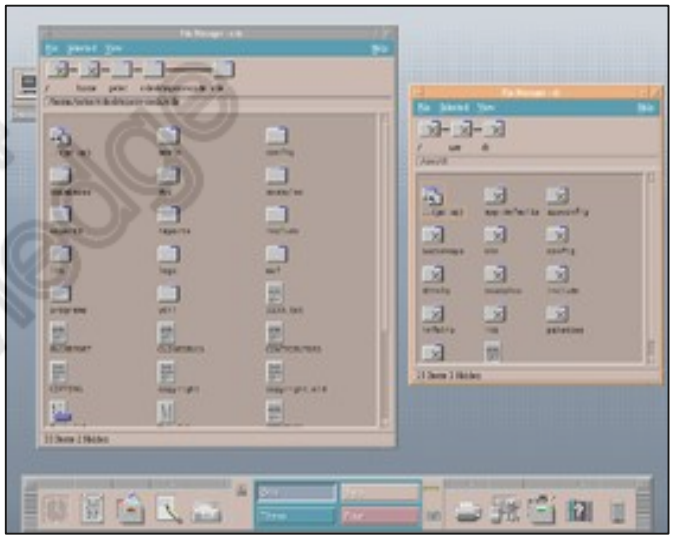
- Environnement intégré 'Desktop' :

. service de connexion	→	fenêtre de login / password
. gestionnaire de fenêtres		
. accessoires	→	émulateur de terminal, horloge, calculette, agenda, éditeur de texte …
. clients élaborés	→	gestionnaire de fichier, outils de documentation, menu d'administration,

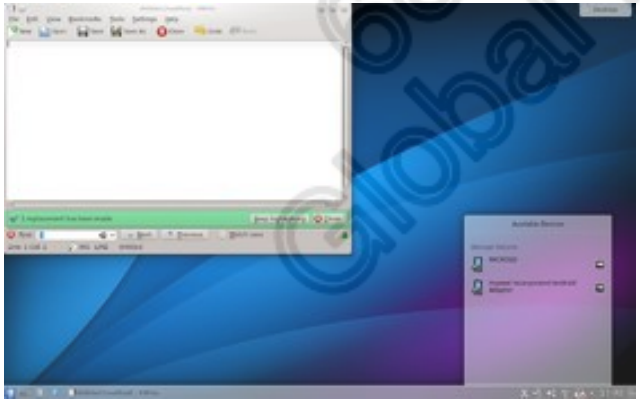
7.4.a

Différents Desktops

- CDE (Common Desktop Environment)
  - → construit par IBM, HP, Novell et Sun, membres de l'Open Group,
  - → déposé sur SourceForge,
  - → puis utilisé par RedHat.



- Environnement Linux :
  - → KDE



Et d'autres …

→ GNOME

7.4.a

Options standards des clients X :

-fg	→	Couleur de premier plan
-bg	→	Couleur d'arrière plan
-title	→	Titre de la fenêtre
-geometry	→	Choix de la taille et de la position
-display	→	Choix du serveur X

Exemples :

```
xterm  -fg yellow  -bg blue  -title "Emulation de terminal"
xterm  -geometry  +100+100
xclock  -geometry  90x80-0+0
```


7.4.b

Choix du serveur X – variable `DISPLAY`

→ définit le serveur X par défaut,

→ évite l'utilisation de l'option standard `-display`

Format :

[host]:server[.screen]

où :

host	→	Nom du système (@ IP) sur lequel s'exécute le serveur X
server	→	Numéro du serveur
screen	→	Numéro d'écran

7.4.c

Global Knowledge

8 – Les bases de la programmation Korn Shell

Procédures …

- Rédaction et exécution d'une procédure (ou script) :

Exécution explicite:

```
$ vi proc
$ ksh proc
```

Exécution implicite:

```
$ vi proc
$ chmod +x proc
$ ./proc
```

→ *début du texte :* `#! Nom_complet_de_l'interpréteur`

exemple : `#! /usr/bin/ksh`

→ *dans les deux cas exécution dans un shell enfant.*

Exécution par le shell courant:

```
$ . ./proc
```

8.1

… et paramètres

- Paramètres automatiques :

$0	→	Nom de la procédure elle-même,
$1, $2, $3 …	→	Valeurs des paramètres reçus,
$*	→	Liste complète des paramètres,
$#	→	Nombre de paramètres,
$?	→	Code retour de la dernière commande appelée,
$$	→	PID courant.

Exemple :

```
#!/usr/bin/ksh
echo  "Je m'appelle $0"
echo  "Nombre d'arguments : $#"
echo  "Liste des arguments : $*"
```

8.1

… exécutions conditionnelles

cmde1 **&&** *cmde2*

→ Lancer *cmde2* si *cmde1* a réussi.

Exemple :

```
$ls -l fic1   &&   cat fic1
```

cmde1 **||** *cmde2*

→ Lancer *cmde2* si *cmde1* a échoué.

Exemple :

```
$ls -l fic1   ||   echo ''il y a un pb ...''
```

8.1

Instructions et contrôles, tests simples : `if - fi`

Syntaxes :

if *cmde*
then
cmde(s)
fi

if *cmde*
then
cmde(s)
else
cmdes(s)
fi

Exemple :

```
if who | grep toto > /dev/null
then
      echo "toto est connecte"
else
      echo "toto n'est pas connecte"
fi
```

8.2.b

… tests séquentiels : `else if`, `elif`

```
if commande
then
commande(s)
else if commande
        then
        commande(s)
        else if commande
                then
                commande(s)
                [else
                commande(s) ]
                fi
        fi
fi
```

```
if commande
then
commande(s)
elif commande
then
commande(s)
elif commande
then
commande(s)
[else
commande(s) ]
fi
```

8.2.c

… commande `test` - syntaxe 1/2 ...

Syntaxe :

test expression
ou
[expression **]**

Opérateurs :

-r *fichier*	→	Droit de lecture sur le fichier
-w *fichier*	→	Droit d'écriture sur le fichier
-x *fichier*	→	Droit d'exécution sur le fichier
-f *fichier*	→	Il s'agit d'un fichier ordinaire
-d *fichier*	→	Il s'agit d'un répertoire
-L *fichier*	→	Il s'agit d'un lien symbolique
-z "*chaine*"	→	La chaîne est de longueur nulle
-n "*chaine*"	→	La chaîne est de longueur non nulle
chaine1 = *chaine2*	→	Les deux chaînes sont identiques
chaine1 != *chaine2*	→	Les deux chaînes sont différentes

8.5.d

… commande `test` – syntaxe 2/2 ...

Opérateur sur les entiers :

entier1 **-eq** *entier2* → Les deux entiers sont égaux

Autres opérateurs sur les entiers :

-ne **-gt** **-ge** **-lt** **-le** → ≠ > ≥ < ≤

Opérateurs logiques :

! Négation

-o OU

-a ET

8.5.d

… commande `test` – exemples 1/2 ...

```
if    test $# -eq 0
then
      echo "Usage : $0 argument" ; exit 1
fi
if    test -f $1
then
      echo "$1 est un fichier ordinaire"
elif  [ -d $1 ]
then
      echo "$1 est un repertoire"
else
      echo "Ni fichier ordinaire, ni repertoire"
fi
```

```
if    test -f $1 -a -x $1
then
      echo "$1 est un fichier ordinaire et executable"
fi
```

8.5.d

… commande `test` – exemple 2/2

```
midi=12
heure=`date '+%H'`
if    test $heure -ge $midi
then
      echo "Il est plus de midi"
else
      echo "Il n'est pas encore midi"
fi
```


8.5.d

Boucle : `for do - done`

Syntaxe :

```
for variable in liste_de_valeurs
do
        cmde(s)
done
```

Exemples :

```
for arg in $*
do
        liste_de_commandes
done
```

Boucler sur tous les paramètres de la procédure

```
for nomfic in *
do
        liste_de_commandes
done
```

Boucler sur tous les noms de fichiers du répertoire courant

```
for i in 1 2 3 4 5
do
        liste_de_commandes
done
```

Faire cinq fois un traitement

8.5.f

Boucles – `while` et `until do - done`

Syntaxes :

while *commande*
do
    cmde(s)
done

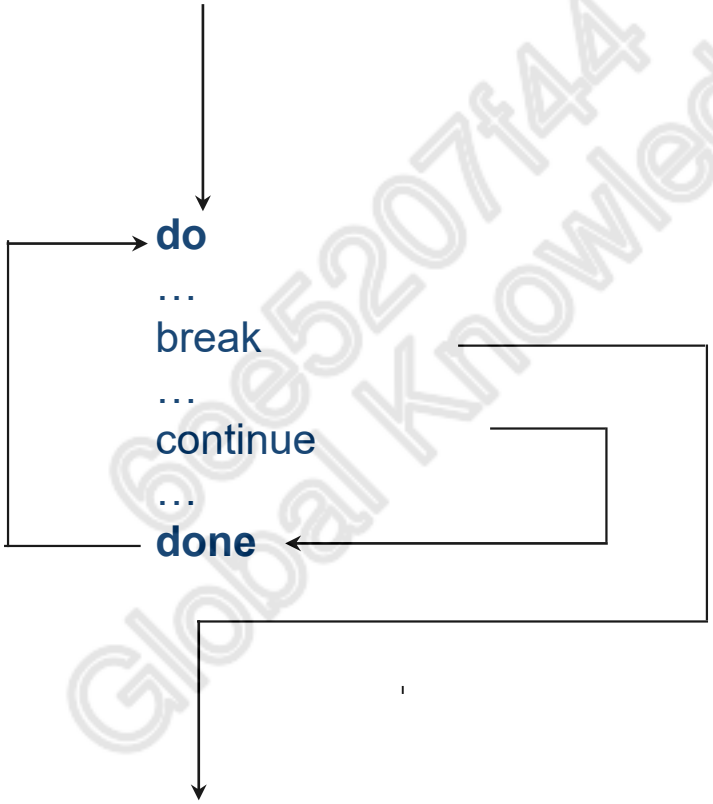
until *commande*
do
    cmde(s)
done

Exemple :

```
who | grep $1 > /dev/null
while test $? -eq 0
do
    echo "$1 est connecte"
    sleep 15
    who | grep $1 > /dev/null
done
echo "$1 n'est pas ou n'est plus connecte"
```


8.5.i

Instructions de branchement : `break`, `continue`



8.5.h

Lecture au clavier - `read`

Syntaxe :

read *liste_de_variables*

Exemples :

```
reponse=o
until test "$reponse" = n
do
     echo "Voulez-vous continuer (o/n) : \c"
     read reponse
done
```

```
reponse=o
until test $reponse = n
do
     echo "Voulez-vous continuer (o/n) : \c"
     read  reponse
     if test -z "$reponse"
     then
          reponse=n
     fi
done
```


8.5.i

Expressions arithmétiques – `let, (( ))`

```
x=10
let x=x+1
```

→ pas d'espace avec la commande `let`

```
(( x = x + 2 ))
(( x = ( x - 2 ) * 3 ))
```

→ les doubles parenthèses permettent les espaces

→ les parenthèses modifient les priorités des opérateurs

```
i=0
while (( i < 100 ))
do
  ...
 let i=i+1
done
```


8.5.g

Expressions arithmétiques, manipulations de chaines de caractères - `expr`

Syntaxe :

expr opérande1 opérateur opérande2
expr expression

Opérateurs : **+ - / * %**

= != < <= > >=

substr length …

Exemples :

```
$ v1=10 ; v2=15
$ expr  $v1 +  $v2
25
$ expr  length  ”bonjour”
7
$
```

Aiguillage : `case` - `esac`

Syntaxe :

```
case valeur in
motif1 )
        cmdes(s) ;;
motif2 )
        cmdes(s) ;;
...
esac
```

| → pour énumération de motif,

* → autre cas, doit être placé en dernier.

Exemple :

```
case  $#  in
     1|2)  echo "On a recu 1 ou 2 arguments"
           ;;
     3)    echo "On a recu 3 arguments"
           ;;
     *)    echo "Mauvais appel, il faut 1 ou 2 ou 3 arguments"
           exit 1
           ;;
esac
echo "Suite du traitement"
```

8.5.i

Fonctions

Syntaxe déclarative :

function *nomFonction*
{
cmde(s)
}

→ s'exécute dans le shell courant,
→ reçoit les arguments $1, $2, …
→ renvoie une valeur $? via l'instruction `return`

Exemple :

```
$function mcd
>{
> cd  $1
> PS1=`pwd` ''> ''
}
$ pwd
/home/stage01
$ mcd ..
/home> mcd
/home/stage01> _
```

8.3.a

Autres commandes internes ':'

. (point) Interpréter la procédure sans créer de shell fils.

: (deux points) Commande 'toujours vraie' (pour des boucles infinies).

```
while  :
do
.........
done
```

Autres commandes : `printf` et caractères de présentation

`printf` en alternative à la commande `echo`.

Accepte les caractères conventionnels suivants :

\a	Signal sonore
\b	Retour arrière
\c	Pas de saut de ligne
\f	Effacement d'écran ou saut de page
\n	Saut de ligne
\r	Retour chariot
\t	Tabulation
\v	Tabulation verticale

… `printf` et caractères de présentation

```
$ printf "\t\t texte précédé de deux <tab>,\n deuxième ligne, puis 2 <rc>.\n\n"
          texte précédé de deux <tab>,
 deuxième ligne, puis 2 <rc>.


$
```

mise en pratique ...

TP11 – Les bases de la programmation shell

5.4.*


Des questions ?