



UNIX

Chapitre 1 : Prise en main et commandes de base

MAIS AU FAIT ... UNIX ... QUI ES-TU ?



Mon nom est UNIX et ma fiche signalétique peut se résumer ainsi  :

● Un système d'exploitation



- interactif,
- multi-utilisateurs,
- multi-tâches,
- multi-langages,

● Un langage de commande

- séquentiel,
- pseudo-parallèle,
- abréviations,
- re-directions d'entrée-sorties,
- commandes de base,
- programmes,
- communications,
- synchronisation...

● Une documentation en ligne

● Des utilitaires

- traitement de texte (TROFF, NROFF)
- gestion d'applications (MAKE )
- gestion de programmes sources (SCCS )...

● Interface sympa...

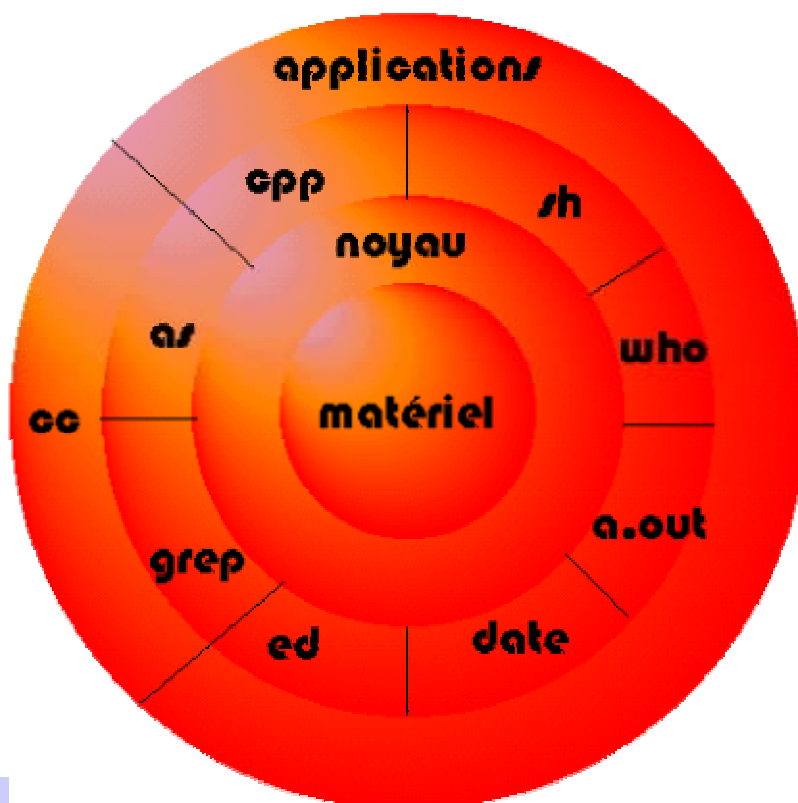
LA STRUCTURE DU SYSTEME



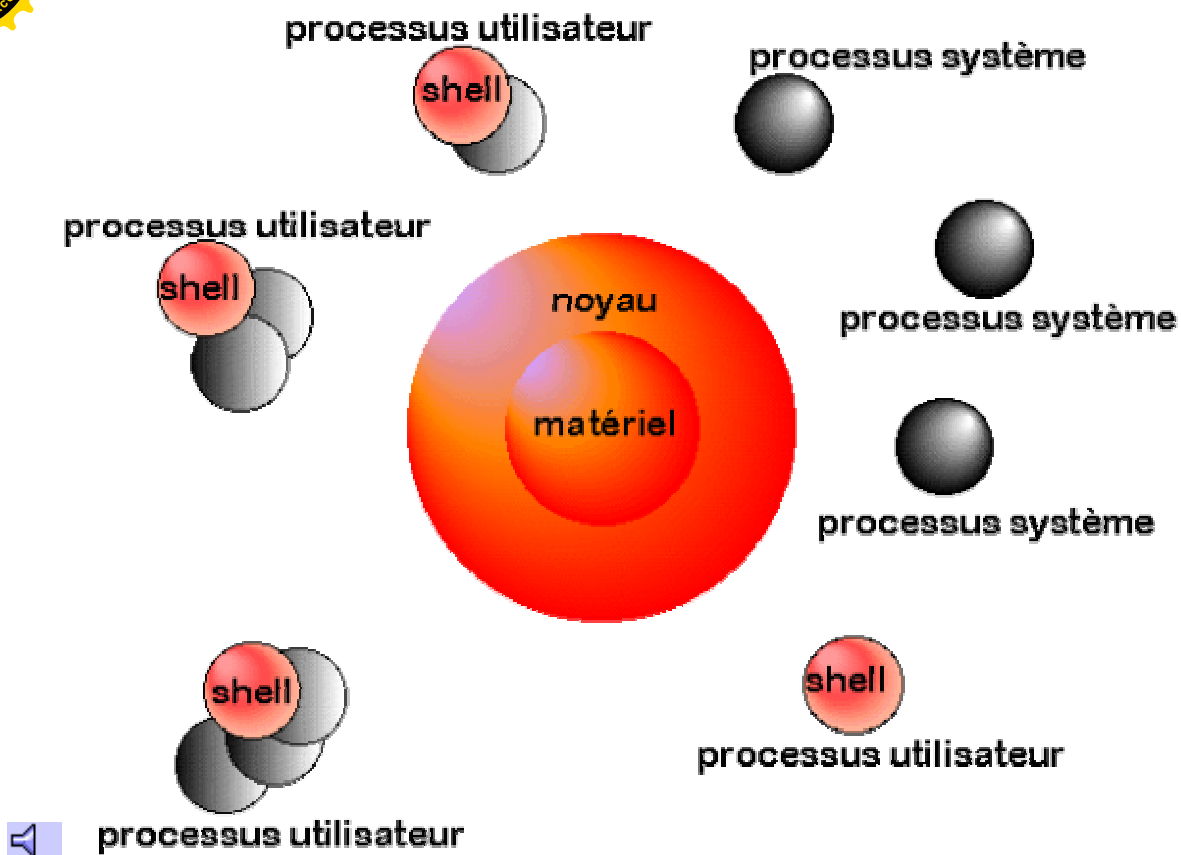
Le système UNIX est constitué de 3 couches principales :

- Le noyau,
- L'interpréteur de commandes Shell, les commandes, ...
- Les utilitaires, les applications.

Le **noyau** offre les services de base ("interface système") pour construire les commandes, les outils, les applications.




- il gère les ressources matérielles (mémoire, unités d'E/S ...), les fichiers, l'allocation de temps UC
- il gère les processus (ex: processus sh)
- il gère la communication et la synchronisation



L'ENVIRONNEMENT UTILISATEUR



L'utilisateur a un ensemble de commandes, d'utilitaires, et d'outils de développement à sa disposition :

- interface utilisateur multi-fenêtres (XWindow, Motif),
- éditeurs de texte ligne (ED), écran (VI, EMACS),
- utilitaires de traitement de texte (TROFF, TeX, ...),
- les commandes de base, les programmes de commandes (scripts), les outils et les applications sont accessibles au travers du langage de commande du système le [SHELL](#) 
- environnement de développement logiciel : compilateurs, debuggers symboliques
...
 - langages C, C++, ADA, Cobol, Pascal, Fortran, Lisp, APL, Prolog ...
- outils de gestion de logiciels
 - gestion de programme,
 - gestion de projets et de versions,
- services de communication (mail, telnet, ftp, WWW, ...),
- outils graphiques de base (GKS, PHIGS),



- outils de bases de données (Oracle, Ingres, Informix, ...),
- applications utilisateurs.

Cette énumération a pour but de vous sensibiliser à une terminologie que vous rencontrerez dans votre vie d'utilisateur Unix.

HISTORIQUE DU SYSTEME UNIX



Un peu d'histoire déjà ancienne et quelques dates : 

- **1968** : Fin du projet MULTICS, étude commune des Bell Labs, General Electric au MIT.
- **1970** : Première version d'UNIX
 - système mono-utilisateur
 - système de gestion de fichiers
 - outils de traitement de textes
 - noyau de système élémentaire
 - interpréteur de commandes élémentaires

Travail initié par Ken THOMPSON

- **1971** : Première version d'UNIX + ...
 - + documentation
 - + plusieurs extensions
 - système de fichiers
 - gestion de processus
 - interface système
 - utilitaires
 - transport sur PDP 11/20

Première version officielle interne UNIX V1 signée Ken THOMPSON et Dennis RITCHIE

- **1972** : UNIX V1 + "pipes" => V2
transport sur PDP 11/20/34/40/45/60/70
- **1973** : UNIX re-écrit en C
BCLP => B => NB => C (Dennis RITCHIE)
- **1974** : UNIX V5 distribuée gratuitement à des universités (UC Berkeley, Columbia U.)
- **1977** : UC. Berkeley (BSD) => CSH, éditeurs, ipc ...



- **1979** : UNIX V7 = V6 + portabilité
- **1980** : ONYX première version pour micros
- **1981** : Première version commercialisée par :
AT&T => System III
UC. Berkeley => BSD 4.1
==> 2 produits indépendants
- **1983** : AT&T => System V
UC. Berkeley => BSD 4.2
- **Aujourd'hui** ...
Intégration des fonctionnalités System V et BSD dans un même standard :
[Posix](#)

AUJOURD'HUI



Le monde informatique s'oriente vers la distribution des ressources et la répartition des traitements.

Normes et standards

- La [norme POSIX](#)
- Vers des normes pour les systèmes ouverts et distribués : [OSF \(Open System Foundation\)](#) et [XOPEN](#) .

Les systèmes distribués

- ré-écriture d'Unix + nouvelles fonctionnalités sur [micro-noyau](#) :
CHORUS ==> Unix International
MACH ==> OSF
- systèmes de fichiers répartis (grand réseau)
AFS (CMU) Andrew File System
- SSI : Single System Image VM répartie

Les environnements multi-fenêtres

- Sunview, XWindow System, Openwindows, Motif
- nouveaux éditeurs (framemaker, Interleaf...)

La sécurité

- Service d'authentification (Client - Serveur)
exemple : [Kerberos](#)

Les systèmes orientés objet

- ré-écriture d'Unix en C++ / micro-noyau
- programmation répartie orientée-objet

Le temps-réel

- reproductibilité des temps d'exécution
- traitement d'événements externes
- ordonnancement des processus (échéances, priorités, préemption / non préemption...)
- performances
- fichiers contigus

COMMENT SE CONNECTER AU SYSTEME UNIX



Le but de ce chapitre est de préciser la procédure de connexion.

Il présente également la manière de quitter le système UNIX.

COMMENT SE CONNECTER

Connexion - Déconnexion

```
login: billy
Password :
$
$ passwd
old passwd:
new passwd:
re-enter new passwd:
$

$ exit
```

D'autres procédures de connexion peuvent être utilisées (notamment pour les interfaces XWindow) qui peuvent ressembler à



Certains systèmes Unix, notamment ceux qui utilisent l'interface XWindow, n'ouvrent pas systématiquement un Shell. L'ouverture d'un Shell se fait en ouvrant une fenêtre terminal.


Des fichiers de démarrage sont exécutés en début de session. Selon l'environnement utilisé, ils se nomment :

```
.login (SOUS Cshell)  
.profile (SOUS Korn Shell, Bourne Shell)
```



Exercice :

Personnalisez votre fichier d'ouverture de session (.login ou .profile selon votre Shell).

Pour connaître le shell avec lequel vous travaillez, tapez la commande `env`  (qui affiche l'ensemble des variables d'environnement et leur valeur) ou la commande `echo $SHELL` (qui affiche la valeur de la variable d'environnement `SHELL`, variable spécifiant le [shell](#) utilisé).



Pour modifier votre fichier d'ouverture de session, utilisez un éditeur de texte à partir de votre environnement de travail.



Vous pouvez par exemple, rajouter la date du jour et votre nom :

```
echo ""  
date  
echo ""  
banner "Je suis untel"  
echo ""
```

Pour tester, après avoir sauvegardé votre modification, tapez la commande `csch .login` dans le cas du Cshell ou `sh .profile` dans le cas du Shell de Bourne, les commandes rajoutées seront alors exécutées.

Remarque sur l'environnement HP :

Sur HPVUE, pour ouvrir une fenêtre terminal (un shell) cliquez sur l'icone  comme le montre la vidéo ,

l'accès à un éditeur de texte se fait en cliquant sur l'icone  comme le montre la vidéo .



UNIX

Chapitre 1 : Prise en main et commandes de base

MAIS AU FAIT ... UNIX ... QUI ES-TU ?



Mon nom est UNIX et ma fiche signalétique peut se résumer ainsi  :

● Un système d'exploitation



- interactif,
- multi-utilisateurs,
- multi-tâches,
- multi-langages,

● Un langage de commande

- séquentiel,
- pseudo-parallèle,
- abréviations,
- re-directions d'entrée-sorties,
- commandes de base,
- programmes,
- communications,
- synchronisation...

● Une documentation en ligne

● Des utilitaires

- traitement de texte (TROFF, NROFF)
- gestion d'applications (MAKE )
- gestion de programmes sources (SCCS )...

● Interface sympa...

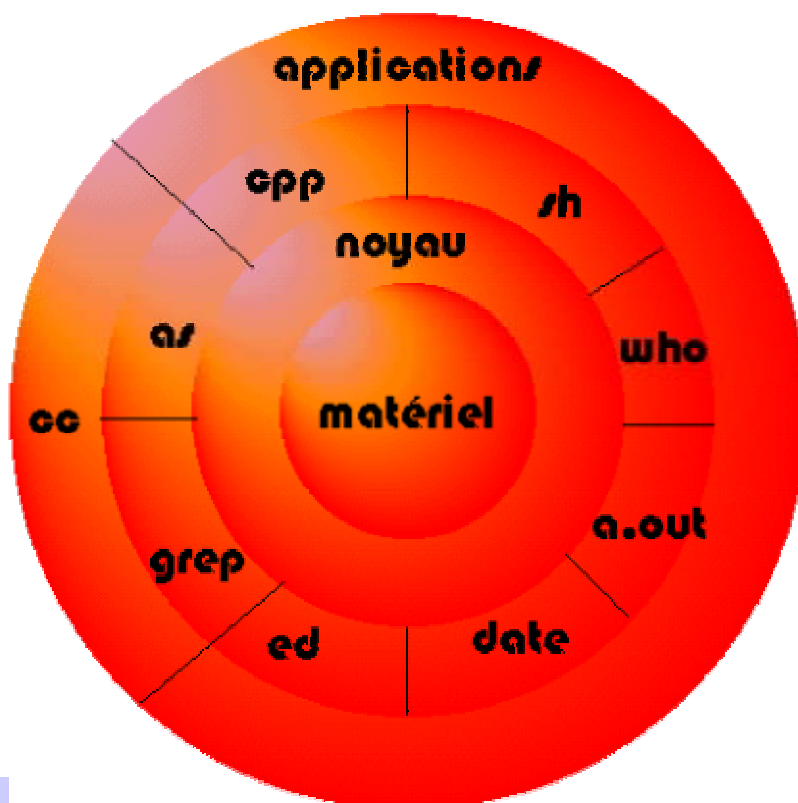
LA STRUCTURE DU SYSTEME



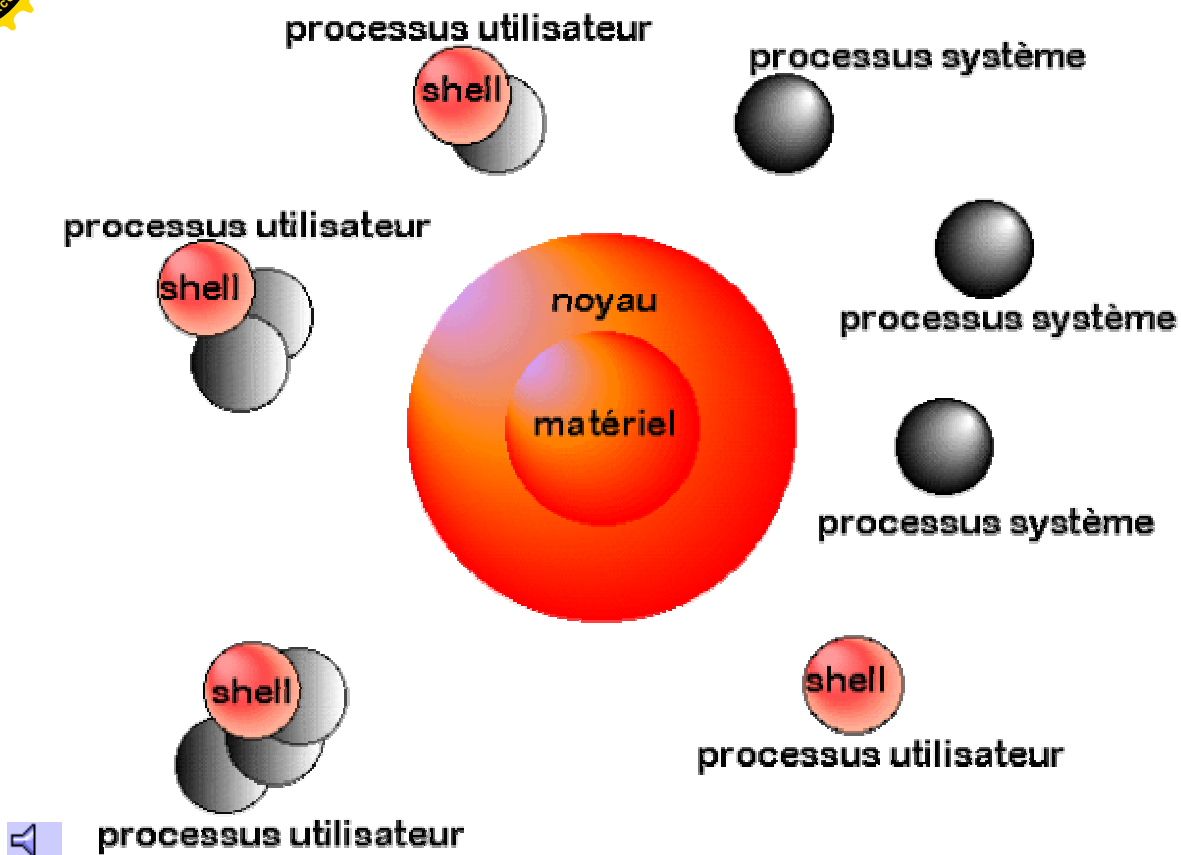
Le système UNIX est constitué de 3 couches principales :

- Le noyau,
- L'interpréteur de commandes Shell, les commandes, ...
- Les utilitaires, les applications.

Le **noyau** offre les services de base ("interface système") pour construire les commandes, les outils, les applications.




- il gère les ressources matérielles (mémoire, unités d'E/S ...), les fichiers, l'allocation de temps UC
- il gère les processus (ex: processus sh)
- il gère la communication et la synchronisation



L'ENVIRONNEMENT UTILISATEUR



L'utilisateur a un ensemble de commandes, d'utilitaires, et d'outils de développement à sa disposition :

- interface utilisateur multi-fenêtres (XWindow, Motif),
- éditeurs de texte ligne (ED), écran (VI, EMACS),
- utilitaires de traitement de texte (TROFF, TeX, ...),
- les commandes de base, les programmes de commandes (scripts), les outils et les applications sont accessibles au travers du langage de commande du système le [SHELL](#) 
- environnement de développement logiciel : compilateurs, debuggers symboliques
...
 - langages C, C++, ADA, Cobol, Pascal, Fortran, Lisp, APL, Prolog ...
- outils de gestion de logiciels
 - gestion de programme,
 - gestion de projets et de versions,
- services de communication (mail, telnet, ftp, WWW, ...),
- outils graphiques de base (GKS, PHIGS),



- outils de bases de données (Oracle, Ingres, Informix, ...),
- applications utilisateurs.

Cette énumération a pour but de vous sensibiliser à une terminologie que vous rencontrerez dans votre vie d'utilisateur Unix.

HISTORIQUE DU SYSTEME UNIX



Un peu d'histoire déjà ancienne et quelques dates : 

- **1968** : Fin du projet MULTICS, étude commune des Bell Labs, General Electric au MIT.
- **1970** : Première version d'UNIX
 - système mono-utilisateur
 - système de gestion de fichiers
 - outils de traitement de textes
 - noyau de système élémentaire
 - interpréteur de commandes élémentaires

Travail initié par Ken THOMPSON

- **1971** : Première version d'UNIX + ...
 - + documentation
 - + plusieurs extensions
 - système de fichiers
 - gestion de processus
 - interface système
 - utilitaires
 - transport sur PDP 11/20

Première version officielle interne UNIX V1 signée Ken THOMPSON et Dennis RITCHIE

- **1972** : UNIX V1 + "pipes" => V2
transport sur PDP 11/20/34/40/45/60/70
- **1973** : UNIX re-écrit en C
BCLP => B => NB => C (Dennis RITCHIE)
- **1974** : UNIX V5 distribuée gratuitement à des universités (UC Berkeley, Columbia U.)
- **1977** : UC. Berkeley (BSD) => CSH, éditeurs, ipc ...



- **1979** : UNIX V7 = V6 + portabilité
- **1980** : ONYX première version pour micros
- **1981** : Première version commercialisée par :
AT&T => System III
UC. Berkeley => BSD 4.1
==> 2 produits indépendants
- **1983** : AT&T => System V
UC. Berkeley => BSD 4.2
- **Aujourd'hui** ...
Intégration des fonctionnalités System V et BSD dans un même standard :
[Posix](#)

AUJOURD'HUI



Le monde informatique s'oriente vers la distribution des ressources et la répartition des traitements.

Normes et standards

- La [norme POSIX](#)
- Vers des normes pour les systèmes ouverts et distribués : [OSF \(Open System Foundation\)](#) et [XOPEN](#) .

Les systèmes distribués

- ré-écriture d'Unix + nouvelles fonctionnalités sur [micro-noyau](#) :
CHORUS ==> Unix International
MACH ==> OSF
- systèmes de fichiers répartis (grand réseau)
AFS (CMU) Andrew File System
- SSI : Single System Image VM répartie

Les environnements multi-fenêtres

- Sunview, XWindow System, Openwindows, Motif
- nouveaux éditeurs (framemaker, Interleaf...)

La sécurité

- Service d'authentification (Client - Serveur)
exemple : [Kerberos](#)

Les systèmes orientés objet

- ré-écriture d'Unix en C++ / micro-noyau
- programmation répartie orientée-objet

Le temps-réel

- reproductibilité des temps d'exécution
- traitement d'événements externes
- ordonnancement des processus (échéances, priorités, préemption / non préemption...)
- performances
- fichiers contigus

COMMENT SE CONNECTER AU SYSTEME UNIX



Le but de ce chapitre est de préciser la procédure de connexion.

Il présente également la manière de quitter le système UNIX.

COMMENT SE CONNECTER

Connexion - Déconnexion

```
login: billy
Password :
$
$ passwd
old passwd:
new passwd:
re-enter new passwd:
$

$ exit
```

D'autres procédures de connexion peuvent être utilisées (notamment pour les interfaces XWindow) qui peuvent ressembler à



Certains systèmes Unix, notamment ceux qui utilisent l'interface XWindow, n'ouvrent pas systématiquement un Shell. L'ouverture d'un Shell se fait en ouvrant une fenêtre terminal.


Des fichiers de démarrage sont exécutés en début de session. Selon l'environnement utilisé, ils se nomment :

```
.login (SOUS Cshell)  
.profile (SOUS Korn Shell, Bourne Shell)
```



Exercice :

Personnalisez votre fichier d'ouverture de session (.login ou .profile selon votre Shell).

Pour connaître le shell avec lequel vous travaillez, tapez la commande `env`  (qui affiche l'ensemble des variables d'environnement et leur valeur) ou la commande `echo $SHELL` (qui affiche la valeur de la variable d'environnement `SHELL`, variable spécifiant le [shell](#) utilisé).



Pour modifier votre fichier d'ouverture de session, utilisez un éditeur de texte à partir de votre environnement de travail.



Vous pouvez par exemple, rajouter la date du jour et votre nom :

```
echo ""  
date  
echo ""  
banner "Je suis untel"  
echo ""
```

Pour tester, après avoir sauvegardé votre modification, tapez la commande `csch .login` dans le cas du Cshell ou `sh .profile` dans le cas du Shell de Bourne, les commandes rajoutées seront alors exécutées.

Remarque sur l'environnement HP :

Sur HPVUE, pour ouvrir une fenêtre terminal (un shell) cliquez sur l'icone  comme le montre la vidéo ,

l'accès à un éditeur de texte se fait en cliquant sur l'icone  comme le montre la vidéo .



UNIX

Chapitre 2 : Les commandes de base du système UNIX



ATTENTION: Pour une meilleure compréhension des notions développées dans ce chapitre, nous vous invitons à le suivre page par page, en commençant par le chapitre "Notion de commande".

NOTION DE COMMANDE



Une commande est une suite de mots séparés par au moins un espace.

Le premier est le nom de la commande, suivi par une liste facultative d' [options](#) et d' [arguments](#).

```
commande [-options] [arg1 ... argn]
```

Les majuscules et les [minuscules](#) sont différenciées dans le système Unix.

Il est possible d'écrire plusieurs commandes sur la même ligne. Le séparateur de commandes est le ";"

Quelques exemples de commandes :

```
ls  
ls -l  
ls -l -a  
ls -la  
ls -al  
ls bidule  
ls -l bidule
```

Remarque : sur la présentation de la syntaxe des commandes, tout texte entre [] (crochets) est optionnel.

Obtenir de l'aide sous Unix

Il existe des [informations](#)  en ligne disponibles sur Unix.

Une manière simple d'obtenir ces informations c'est d'utiliser la commande `man`

■ Pour obtenir la correspondance des codes ASCII en octal et hexadécimal, tapez la commande : `man ascii`

■ Pour avoir la syntaxe de la commande `man`, tapez : `man man`

La commande `man`




Définition

Cette commande donne accès au manuel en ligne du système en vue d'obtenir de la documentation.



Syntaxe

`man` [[section](#) ] *nom de la commande*



Exercice

- Affichez le manuel de la commande `ls`
- Affichez le manuel de la commande `whoami`
- Tapez :

```
man intro
```

- Puis :

```
man 2 intro
```

- et enfin :

```
man 3c intro
```


LES COMMANDES DE GESTION DE FICHIERS



Dans ce chapitre, nous allons aborder la notion de système de fichiers, pour cela nous donnerons la définition d'un fichier, les différents types de fichiers existant sous Unix ainsi que les chemins d'accès.

Nous détaillerons aussi les commandes de gestion de fichiers et de catalogues.

Le chapitre se divise en :

- la définition d'un système de fichiers,
- les principales commandes de manipulation de fichiers,
- les principales commandes de manipulation de répertoires.

LE SYSTEME DE FICHIERS



■ Qu'est-ce qu'un fichier ?

Un fichier UNIX est une suite de caractères non structurée. UNIX n'a pas la notion d'organisation de fichier (indexée, relative, etc ...).

A tout fichier est attribué un bloc d'informations appelé noeud d'index ou i-node. Cet i-node contient des informations générales concernant le fichier:

- sa taille (en octets),
- l'adresse des blocs utilisés par le fichier sur le disque,
- l'identification du propriétaire du fichier,
- les droits d'accès des différents groupes d'utilisateurs,
- le type du fichier,
- un compteur de liens,
- les dates des principales opérations (création, mise à jour, consultation).

Remarque: le i-node ne contient pas le nom du fichier.

■ Les types de fichiers

Dans le système UNIX il existe 3 types de fichiers:

- Les **fichiers ordinaires** peuvent être :
 - des programmes exécutables (compilateurs, éditeurs, tableurs, ...)



- des fichiers texte
- des fichiers de données

Il n'y a pas de format à respecter pour le nom des fichiers UNIX (jusqu'à 256 caractères).

- **Les fichiers spéciaux :**

Ce sont des fichiers associés à un dispositif d'entrée/sortie (E/S) physique. Ils sont traités par le système comme des fichiers ordinaires mais la lecture et l'écriture sur ces fichiers activent les mécanismes physiques associés (drivers).

Il existe 2 types de fichiers spéciaux:

- mode caractère : E/S réalisées caractère par caractère (terminaux, imprimantes, lignes de communication, ...)
- mode bloc : E/S réalisées par blocs de caractères (disques, bandes).

- **Les répertoires :**

Contiennent les couples (i-node, nom de fichier). On ne peut créer, effacer, lire ou écrire dans des répertoires qu'au moyen de primitives systèmes spécifiques.

Les répertoires sont aussi appelés catalogues ou directories.

- **Conventions de nommage des répertoires :**

- . (point) désigne le répertoire courant.
- .. (point point) désigne le répertoire père du répertoire courant.
- / (slash) désigne la racine de l'arborescence des fichiers.
Dans la désignation d'un chemin, c'est un séparateur de catalogue.
- ~ (tilde) désigne le "home directory" de l'utilisateur.

- **Chemin d'accès à un fichier ou à un répertoire :**

Le chemin d'accès à un fichier (ou à un catalogue) est la description qui permet d'identifier le fichier (ou le catalogue) dans la hiérarchie du système.

Le chemin d'accès correspond en une suite de noms de répertoires séparés par des caractères / (slash) et terminé par le nom du fichier ou du répertoire.

Ainsi le chemin d'accès suivant :

```
/users/fudmip/prof/.login
```

représente le fichier `.login` qui se trouve dans le catalogue `prof` [catalogue de connexion](#) lui-même placé sous le catalogue `fudmip`, lui-même contenu dans le catalogue `users` qui se trouve sous la racine `/`.

Remarque : Le caractère `/` marque la séparation entre catalogues lorsqu'on décrit le "chemin d'accès" à un fichier ou un catalogue.

CHEMIN D'ACCES A UN FICHIER

Il y a deux façons de spécifier le chemin d'accès (`pathname` en anglais) à un fichier :

- soit on décrit le chemin à partir de la racine (`/`) c'est donc un [chemin d'accès absolu](#) comme vu dans la page précédente

Exemple:

```
/usr/lib/libc.a  
/etc
```

- soit on décrit le chemin à partir du [catalogue courant](#), c'est [un chemin relatif](#) (il ne commence donc pas par un `/`).

MANIPULATION DE FICHIERS



AFFICHER LA LISTE DES FICHIERS D'UN REPERTOIRE

La commande `ls` affiche la liste des fichiers du répertoire courant :

```
$ ls  
boite  fic1   fic2   Fich.c  
$
```

La commande `ls` affiche la liste des fichiers du répertoire passé en paramètre (`/tmp`) :

```
$ ls /tmp  
env      input.tif  Ktal      last_uuid  log  
$
```

● L'option `-l` affiche toutes les informations sur les fichiers :

```
$ ls -l /tmp
total 80
-rw-rw-rw- 1 www www 4905 janv 18 11:17 env
-rw-r--r-- 1 ubanell fudmip 32587 janv 17 14:25 input.tif
drwxrwxrwx 3 baque fudmip 1024 janv 17 11:24 Ktal
-rw-rw-rw- 1 ubanell fudmip 16 janv 18 17:40 last_uuid
-rw-rw-rw- 1 www www 53 janv 17 16:45 log
$
```

● L'option `-la` permet aussi d'afficher la liste des fichiers invisibles  :

```
$ ls -la
total 14
drwxrwxr-x 3 ubanell fudmip 1024 janv 18 17:52 boite
-rw-rw-r-- 1 ubanell fudmip 7 janv 18 17:45 fic1
-rw-rw-r-- 1 ubanell fudmip 9 janv 18 17:45 fic2
-rw-r--r-- 1 ubanell fudmip 391 janv 12 11:29 Fich.c
drwxr-xr-x 3 ubanell fudmip 1024 janv 18 18:01 .
-rw-rw-r-- 1 ubanell fudmip 0 janv 18 18:01 .ma-config
drwxrwxrwx 32 ubanell fudmip 2048 janv 18 17:47 ..
$
```

● `-R` permet de lister toute la sous-hiérarchie :

```
$ ls -lR
total 8
drwxrwxr-x 3 ubanell fudmip 1024 janv 18 17:52 boite
-rw-rw-r-- 1 ubanell fudmip 7 janv 18 17:45 fic1
-rw-rw-r-- 1 ubanell fudmip 9 janv 18 17:45 fic2
-rw-r--r-- 1 ubanell fudmip 391 janv 12 11:29 Fich.c

./boite:
total 8
-rw-rw-r-- 1 ubanell fudmip 5 janv 18 17:52 essai
-rw-r--r-- 1 ubanell fudmip 152 janv 15 11:22 Fic4
-rw-r--r-- 1 ubanell fudmip 152 janv 15 11:22 Fich
drwxrwxr-x 2 ubanell fudmip 1024 janv 15 11:23 Katal

./boite/Katal:
total 0
-rw-rw-r-- 1 ubanell fudmip 0 janv 15 11:23 pp
$
```

 La commande `ls` 



Définition

Cette commande affiche les caractéristiques  des fichiers contenus dans un catalogue.



Syntaxe

```
ls [-options] [fichiers ou répertoires]
```



Exercice

De nombreuses options sont disponibles.

- Testez la commande `ls /tmp` : vous visualisez ainsi le contenu du répertoire `/tmp`
- Testez ensuite la commande `ls -l /tmp` qui permet de lister tous les fichiers de façon complète avec les modes, les groupes, les créateurs et les dates.
- Testez la commande `ls -al` et constatez que les fichiers cachés (commençant par un point) apparaissent cette fois-ci.
- Testez enfin la commande `ls -li /tmp` et constatez que l'inode de chaque fichier est affiché.

AFFICHER LE CONTENU D'UN FICHIER

● Affichage du contenu du fichier `.profile` : `cat .profile`

```
$ cat .profile

# @(#) $Revision: 66.1 $

# Default user .profile file (/bin/sh initialization).

# Set up the terminal:
eval ' tset -s -Q -m ' :?hp' '
stty erase "^H" kill "^U" intr "^C" eof "^D"
stty hupcl ixon ixoff
tabs

# Set up the search paths:
PATH=$PATH:.

# Set up the shell environment:
set -u
trap "echo 'logout'" 0


# Set up the shell variables:
EDITOR=vi
export EDITOR

$
```

La commande cat

Définition

Cette commande permet d'afficher le contenu d'un fichier dont le nom est passé en paramètre.

En réalité la commande cat concatène le contenu de tous les fichiers passés en arguments et envoie le résultat sur l'[écran](#) 

Elle permet aussi de créer un fichier en utilisant la redirection d'E/S (>).

Syntaxe

```
cat essai.c
```

```
cat > truc
```

Exercices

1) Afficher le contenu d'un fichier :

- [Visualisez](#) le contenu de votre répertoire courant
- Choisissez un fichier et visualisez le avec la commande **cat**

2) Créer un fichier :

- Créez un fichier texte nommé *toto* dans votre répertoire : `cat > toto`
- Saisissez un texte
- Pour terminer votre fichier, taper [^D](#) en début de ligne .
- Votre fichier *toto* est maintenant créé, pour vous en assurer, vous pouvez le visualiser en tapant la commande `cat toto`

AFFICHER LE CONTENU D'UN FICHIER AVEC ARRÊT A CHAQUE PAGE

● Affichage du contenu du fichier `.profile` page par page : `more .profile`

```
$ more .profile

# @(#) $Revision: 66.1 $

# Default user .profile file (/bin/sh initialization).

# Set up the terminal:
    eval ' tset -s -Q -m ' :?hp' '
    stty erase "^H" kill "^U" intr "^C" eof "^D"
    stty hupcl ixon ixoff
    tabs

# Set up the search paths:
    PATH=$PATH:.

--Encore--(67%)
```

La commande `more`

Définition

Cette commande permet d'afficher le contenu d'un fichier avec arrêt à chaque page. On peut alors utiliser :

- la touche **ESPACE** pour passer à la page suivante,
- la touche **h** pour avoir l'aide,
- la touche **q** ou **^D** pour sortir de `more`.

Remarque : la commande `more` fait partie de la famille des [pagers](#) , il en existe d'autres `pg`, `less`, ...

Syntaxe


`more mon-fichier`



Exercice

- Visualisez le contenu de votre répertoire courant
- Choisissez un fichier et visualisez le avec la commande `more` en testant les différentes options qui sont à votre disposition en tapant `h`.

AFFICHAGE PARTIEL DU CONTENU D'UN FICHIER

Les commandes suivantes permettent d'afficher [à l'écran](#)  une partie du contenu d'un fichier :

- les premières lignes : [head](#)
- les dernières lignes : [tail](#)
- certaines lignes : [grep](#)
- des parties de lignes : [cut](#)

LES PREMIERES LIGNES D'UN FICHIER



Voici le contenu du fichier **Fich**

```
$ cat Fich
aaaaaaaaaa
bbbbbbbbbbbbbbbb
cccccccccccccc
dddddddddddddddd
eeeeeeeeeeeeeeee
ffffff
gggggggggggggggg
hhhhhhhhhhhhhhh
iiiiiiiiiiiiiii
jjjjjjjjjjjjjjj
kkkkkkk
$
```

La commande **head** affiche les 10 premières lignes du fichier passé en paramètre:


```
$ head Fich
aaaaaaaaaaaa
bbbbbbbbbbbbbbbb
cccccccccccccc
dddddddddddddddd
eeeeeeeeeeeeeeee
ffffff
gggggggggggggggg
hhhhhhhhhhhhhhh
iiiiiiiiiiiiiii
jjjjjjjjjjjjjjj
$
```

● L'option `-2` permet d'afficher les deux premières lignes

```
$ head -2 Fich
aaaaaaaaaaaa
bbbbbbbbbbbbbbbb
$
```

La commande head

Définition

Cette commande affiche les premières lignes d'un fichier.
Des options permettent de modifier le nombre de lignes à afficher.

Syntaxe

```
head [-n] [fichier1] [fichier2 ...]
```

Exercice

- Créez un fichier nommé `bidule` de 10 lignes
- Affichez les 3 premières lignes de `bidule`

LES DERNIERES LIGNES D'UN FICHIER



Voici le contenu du fichier **Fich**

```
$ cat Fich
aaaaaaaaaaaa
bbbbbbbbbbbbbbbb
cccccccccccccc
dddddddddddddddd
eeeeeeeeeeeeeeee
ffffff
gggggggggggggggg
hhhhhhhhhhhhhhh
iiiiiiiiiiiiiiii
jjjjjjjjjjjjjjj
kkkkkkk
$
```

La commande `tail` affiche les 10 dernières lignes du fichier passé en paramètre.

```
$ tail Fich
bbbbbbbbbbbbbbbb
cccccccccccccc
dddddddddddddddd
eeeeeeeeeeeeeeee
ffffff
gggggggggggggggg
hhhhhhhhhhhhhhh
iiiiiiiiiiiiiiii
jjjjjjjjjjjjjjj
kkkkkkk
$
```

L'option `-2` permet d'afficher les 2 dernières lignes


```
$ tail -2 Fich
jjjjjjjjjjjjjjj
kkkkkkk
$
```

L'option `+5` affiche de la cinquième à la dernière ligne

```
$ tail +5 Fich
eeeeeeeeeeeeeeee
ffffff
BBBBBBBBBBBBBBBB
hhhhhhhhhhhhhhh
iiiiiiiiiiiiiii
jjjjjjjjjjjjjjj
kkkkkkk
$
```

La commande tail

Définition

Cette commande affiche les dernières lignes  d'un fichier.
Des options permettent d'en modifier le nombre par défaut.

Syntaxe

```
tail [+/-n] [fichier]
```

Exercice

- Créez un fichier nommé `bidule` de 10 lignes
- Affichez les 5 dernières lignes de `bidule`

CERTAINES LIGNES D'UN FICHIER



● Voici le contenu du fichier **Fich.c**:

```
$ cat Fich.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define TAILLE 512
#define LF 10

main(argc,argv)
int argc;
char *argv[];
{
FILE *fpt;
char *chaine;
int nbre=0;

chaine=(char *) malloc(sizeof(char) * 10);
if (fpt = fopen(argv[1],"r"))
    {fgets(chaine,10,fpt);
    nbre=atoi(chaine);
    fclose(fpt);}
nbre++;
fpt=fopen(argv[1],"w");
fprintf(fpt,"%d",nbre);
fclose(fpt);return(nbre);
}
```

● La commande `grep` affiche toutes les lignes contenant le mot `define` du fichier `Fich.c`

```
$ grep define Fich.c
#define TAILLE 512
#define LF 10
$
```

● L'option `-c` permet d'afficher le nombre de lignes contenant le mot `define`

```
$ grep -c define Fich.c
2
$
```

● Les options `v` et `n` permettent respectivement d'afficher toutes les lignes **ne contenant pas** la chaîne de caractères recherchée ainsi que le **numéro de ligne**. Dans l'exemple suivant nous recherchons les lignes ne contenant pas le caractère [ESPACE](#).

```
$ grep -nv " " Fich.c
6:
7:main(argc,argv)
10:{
14:
17:    {fgets(chaine,10,fpt);
18:    nbre=atoi(chaine);
19:    fclose(fpt);}
20:nbre++;
21:fpt=fopen(argv[1],"w");
22:fprintf(fpt,"%d",nbre);
23:fclose(fpt);return(nbre);
24:}
$
```

La commande grep



Définition

La commande **grep** affiche toutes les lignes d'un fichier contenant la chaîne de caractères spécifiée en argument.

Il est possible d'utiliser des [métacaractères](#)  pour définir la chaîne à rechercher.



Syntaxe

La syntaxe de base est :

```
grep chaîne fichier
```



Exercice

- Affichez la ligne du fichier `/etc/passwd` qui contient votre *nom de login*.

```
grep $USER /etc/passwd
```

ou

```
grep $LOGNAME /etc/passwd
```

selon la variable disponible sur votre système.

Pour connaître l'ensemble des variables d'environnement de votre système exécutez la commande `env`

UNE PARTIE DES LIGNES D'UN FICHIER



Voici le contenu du fichier **passwd** :

```
$ cat passwd
adm:*:4:4::/usr/adm:/bin/sh
lp:*:9:7::/usr/spool/lp:/bin/sh
hpdb:*:27:1:ALLBASE:/:/bin/sh
nobody:*:-2:60001:uid nobody:/:
untel:P1rMrwmlpxhzI:201:200::/users/untel:/bin/sh
lui:Rqhhggqxq5wk2:203:200::/users/lui:/bin/tcsh
elle:tfghnb3wrverc:204:200::/users/elle:/bin/csh
moi:PscrgZjb1bn3I:0:0::/users/moi:/bin/tcsh
$
```

La commande `cut` avec l'option `-c2-5` affiche les caractères du 2^{ième} au 5^{ième} de toutes les lignes

```
$ cut -c2-5 passwd
dm:*
p:*:
pdb:
obod
ntel
ui:R
lle:
oi:P
$
```

Pour afficher les six premiers caractères et le 10^{ième} de toutes les lignes :

```
$ cut -c-6,10 passwd
adm:*::
lp:*:9:
hpdb:*:
nobody-
untel:M
lui:Rqg
elle:tn
moi:PsZ
$
```

Un caractère séparateur peut délimiter des champs dans une ligne (: par exemple)
`cut` peut alors sélectionner certains champs dans toutes les lignes du fichier.

L'option -f permet de déterminer le champ à sélectionner.

L'option -d permet de spécifier le caractère séparateur de champ.

L'exemple suivant sélectionne les champs 1 et 7 du fichier `passwd`, le séparateur de champ est ici `:`.

```
$ cut -f1,7 -d: passwd
adm:/bin/sh
lp:/bin/sh
hpdb:/bin/sh
nobody:
untel:/bin/sh
lui:/bin/tcsh
elle:/bin/csh
moi:/bin/tcsh
$
```

La commande cut

Définition

Cette commande permet d'afficher une partie de chaque ligne du fichier passé en paramètre.

Remarque

: `grep` affiche tout le contenu de certaines lignes,

`cut` affiche une certaine partie de toutes les lignes, comme indiqué dans l'[exemple](#)

Syntaxe

```
cut -cliste fichier1 [fichier2 ...]
```

```
cut -fliste [-dcaractère] fichier1 [fichier2 ...]
```

-cliste liste d'entiers ou intervalles indiquant les caractères à afficher

-dcaractère précise le séparateur de *champ*

-fliste liste d'entiers précisant le n° des *champs* à afficher

Exercice

- Affichez les 10 premiers caractères de chaque ligne du fichier `/etc/passwd`
- Affichez les champs **1** et **5** du fichier `/etc/passwd`

COPIER UN FICHIER



■ Dans le catalogue courant, voici le contenu du fichier `F`

```
$ ls -l
total 4
drwxrwxr-x  2 ubanell  fudmip      24 janv 12 16:50 bricabrac
-rw-rw-r--  1 ubanell  fudmip      60 janv 12 16:34 F
$ cat F
Voici la premiere ligne de mon fichier
et la deuxieme ligne
$
```

■ La commande `cp` copie le fichier `F` dans un fichier que l'on nomme `T`.

```
$ cp F T
$ ls -l
total 6
drwxrwxr-x  2 ubanell  fudmip      24 janv 12 16:50 bricabrac
-rw-rw-r--  1 ubanell  fudmip      60 janv 12 16:34 F
-rw-rw-r--  1 ubanell  fudmip      60 janv 12 16:51 T
$ cat T
Voici la premiere ligne de mon fichier
et la deuxieme ligne
$
```

■ L'option `-R` permet de copier toute une hiérarchie (`bricabrac`) sur un catalogue que l'on précise en argument (`boite`).

```
$ cp -R bricabrac boite
$ ls -l
total 8
drwxrwxr-x  2 ubanell  fudmip      24 janv 12 16:56 boite
drwxrwxr-x  2 ubanell  fudmip      24 janv 12 16:50 bricabrac
-rw-rw-r--  1 ubanell  fudmip      60 janv 12 16:34 F
-rw-rw-r--  1 ubanell  fudmip      60 janv 12 16:51 T
$
```



La commande `cp`





Définition

Cette commande permet de copier des fichiers.

L'option -R autorise la copie de catalogue.



Syntaxe

```
cp  fic-source  fic-cible
cp  fic-source  ktal-cible
cp -R ktal-source ktal-cible
```



Exercices

- Créez un fichier `ii` de quelques lignes
- Copiez ce fichier dans `bidule`

DETRUIRE UN FICHIER



La commande `rm` supprime le fichier passé en paramètre (`fic1` dans l'exemple).

```
$ rm Fic1
$
```

L'option -i demande confirmation de la destruction.

```
$ rm -i Fic1
Fic1: ? (o/n) o
$
```

Il est possible de spécifier plusieurs fichiers à détruire.

L'exemple suivant efface 3 fichiers (`Fic1`, `Fic2`, `Fic3`) en demandant confirmation:

```
$ rm -i Fic2 Fic3 Fic4
Fic2: ? (o/n) o
Fic3: ? (o/n) n
Fic4: ? (o/n) o
$
```

L'option -R permet d'effacer un catalogue et toute son arborescence:

```
$ ls -lR
total 6
drwxrwxr-x   2 ubane11  fudmip      1024 janv 15 10:31 boite
-rw-rw-r--   1 ubane11  fudmip         0 janv 15 10:18 Fic3
-rw-r--r--   1 ubane11  fudmip      152 janv 12 09:41 Fich
-rw-r--r--   1 ubane11  fudmip      391 janv 12 11:29 Fich.c

./boite:
total 2
-rw-r--r--   1 ubane11  fudmip      391 janv 15 10:29 machin
-rw-rw-r--   1 ubane11  fudmip         0 janv 15 10:29 truc
$ rm -R boite
$ ls -lR
total 4
-rw-rw-r--   1 ubane11  fudmip         0 janv 15 10:18 Fic3
-rw-r--r--   1 ubane11  fudmip      152 janv 12 09:41 Fich
-rw-r--r--   1 ubane11  fudmip      391 janv 12 11:29 Fich.c
$
```

La commande rm

Définition

Cette commande permet de détruire les fichiers passés en paramètres.

De nombreuses options de cette commande sont pratiques mais **dangereuses** !

Par défaut, la commande `rm` ne demande aucune confirmation : les fichiers sont donc irrémédiablement perdus.

Syntaxe

```
rm -[Rfi] mon-fichier
rm -i mon-fichier
rm -f mon-fichier
rm -R mon-fichier
```

Exercices

- Créez les fichiers `ii`, `jj` et `truc` à l'aide de la commande: `touch ii jj truc`.
- Effacez le fichier `truc`.
- Effacez le fichier `ii` et `jj` en demandant confirmation.

RENOMMER OU DEPLACER UN FICHIER



Voici le contenu du catalogue:

```
$ ls -lR
total 14
drwxrwxr-x  2 ubanell  fudmip      24 janv 15 11:19 boite
-rw-r--r--  1 ubanell  fudmip     152 janv 12 09:41 Fic
-rw-r--r--  1 ubanell  fudmip     152 janv 15 11:22 Fic1
-rw-r--r--  1 ubanell  fudmip     152 janv 15 11:22 Fic3
-rw-r--r--  1 ubanell  fudmip     152 janv 15 11:22 Fic4
-rw-r--r--  1 ubanell  fudmip     391 janv 12 11:29 Fich.c
drwxrwxr-x  2 ubanell  fudmip    1024 janv 15 11:23 Katal

./boite:
total 0

./Katal:
total 0
-rw-rw-r--  1 ubanell  fudmip        0 janv 15 11:23 pp
$
```

La commande `mv` permet de changer le nom du fichier `Fic` en `Fic.old`:

```
$ mv Fic Fic.old
$
```

L'option `-i` demande confirmation:

```
$ mv -i Fic1 Fic.old
supprimer Fic.old? (o/n) o
$
```

Déplacement du fichier `Fic3` dans le répertoire `boite` en changeant son nom en `Fic2`:

```
$ mv Fic3 boite/Fic2
$
```

Déplacement du fichier `Fic4` dans le répertoire `boite` sans changer son nom:

```
$ mv Fic4 boite
$
```

Déplacement ou renommage du répertoire `Ktal` dans le répertoire `boite`:

\$ mv Katal boite

● Affichage du résultat:

```
$ ls -lR
total 4
drwxrwxr-x   3 ubanell  fudmip      1024 janv 15 11:39 boite
-rw-r--r--   1 ubanell  fudmip        391 janv 12 11:29 Fich.c
-rw-rw-r--   1 ubanell  fudmip         0 janv 15 11:35 Fic.old
-rw-rw-r--   1 ubanell  fudmip         0 janv 15 11:33 ii

./boite:
total 6
-rw-r--r--   1 ubanell  fudmip      152 janv 15 11:22 Fic2
-rw-r--r--   1 ubanell  fudmip      152 janv 15 11:22 Fic4
drwxrwxr-x   2 ubanell  fudmip     1024 janv 15 11:23 Katal

./boite/Katal:
total 0
-rw-rw-r--   1 ubanell  fudmip         0 janv 15 11:23 pp
$
```

La commande mv

Définition

Cette commande permet de déplacer ou de renommer un fichier ou un répertoire.

Syntaxe

Renommage d'un fichier :

`mv mon-fichier nouv-fichier`

Exercices

- Créez les fichiers `un`, `deux` et `trois` à l'aide de la commande: [touch](#) `un` `deux` `trois`.
- Renommez `un` en `1`.
- Renommez `deux` en `1`.

Vous avez perdu le contenu de l'ancien `un` vous auriez du utiliser `-i` qui aurait demandé confirmation avant d'écraser.

- Déplacez le fichier `trois` dans le catalogue `/tmp`.

COMPARAISON DE 2 FICHIERS

UNIX met a disposition 2 commandes pour comparer le contenu de fichiers :

- [`cmp`](#) indique si les contenus des fichiers sont les mêmes.
- [`diff`](#) affiche les modifications à apporter pour les rendre identiques.

COMPARAISON DU CONTENU DE 2 FICHIERS



La commande `cmp` compare le contenu des 2 fichiers passés en paramètre (`ii` et `jj`) :

```
$ cat ii
1er ligne
2eme ligne
3eme ligne
$ cat jj
1er ligne
seconde ligne
3eme ligne
$ cmp ii jj
différence entre ii et jj: caractère 11, ligne 2
$
```

Constatez que `ii` et `ll` sont identiques: :

```
$ cp ii ll
$ cmp ii ll
$
```



La commande `cmp`



Définition

Cette commande permet de comparer le contenu de 2 fichiers.
Elle affiche le numéro de ligne et de caractère de la **première** différence rencontrée.



Syntaxe

```
cmp fichier1 fichier2
```



Exercices

- Créez 2 fichiers, `essai1` et `essai2` pratiquement identiques,
- Comparez les 2 fichiers
- Modifiez l'un des 2 fichiers avec les indications renvoyées par la commande `cmp` pour les rendre identiques
- Comparez les 2 fichiers

DIFFERENCE ENTRE 2 FICHIERS



Voici les contenus des fichiers `ii` et `jj`

```
$ cat ii
1er ligne
2eme ligne
3eme ligne
4eme ligne
$ cat jj
1er ligne
seconde ligne
3eme ligne
quatrieme ligne
5eme ligne
$
```

La commande `diff` affiche à l'écran les différences de contenu entre `ii` et `jj`:

```
$ diff ii jj
2c2
< 2eme ligne
---
> seconde ligne
4c4,5
< 4eme ligne
---
> quatrieme ligne
> 5eme ligne
$
```

La commande diff

Définition

Cette commande retourne les [modifications à apporter](#) au premier fichier passé en paramètre pour le rendre identique au second.

Le résultat de la commande `diff` est affiché suivant la syntaxe:

```
«N° de lignes dans le 1er fichier» « action à effectuer » «N° de lignes
dans le 2ème fichier»

< contenu du 1er fichier
---
> contenu du 2ème fichier
```

La commande `diff` est souvent associée avec la commande [patch](#).

Syntaxe

```
diff [-e] fichier1 fichier2
```

Exercice

- Créez 2 fichiers pratiquement identiques
- Affichez les différences entre les 2 fichiers
- relancez la commande avec l'option `-e` pour constater que la syntaxe du résultat est différente.

TRIER DES FICHIERS



Voici le contenu du fichier `clients`

```
$ cat clients
M:DUPOND:Gerard:38:Toulouse
F:DURAND:Claude:25:Auch
M:ABADIE:Marcel:42:Lannemezan
M:ESCLOP:Auguste:17:Aubin
F:DURANT:Sophie:24:Albi
M:DURAND:Bernard:33:Tournay
$
```

La commande `sort` permet de trier le fichier `clients` par ordre alphabétique:

```
$ sort clients
F:DURAND:Claude:25:Auch
F:DURANT:Sophie:24:Albi
M:ABADIE:Marcel:42:Lannemezan
M:DUPOND:Gerard:38:Toulouse
M:DURAND:Bernard:33:Tournay
M:ESCLOP:Auguste:17:Aubin
$
```

L'option `+` permet de spécifier le n° du champ à partir duquel le tri doit s'effectuer.

L'option `-t` permet de spécifier le séparateur de champ qui est ici «:».

```
$ sort -t: +1 clients
M:ABADIE:Marcel:42:Lannemezan
M:DUPOND:Gerard:38:Toulouse
M:DURAND:Bernard:33:Tournay
F:DURAND:Claude:25:Auch
F:DURANT:Sophie:24:Albi
M:ESCLOP:Auguste:17:Aubin
$
```

Remarque : La numérotation des champs commence à 0 (le tri porte ici sur le 2^{ème} champ).

Il est possible de limiter le tri sur quelques champs consécutifs.

L'option `-` permet de spécifier le dernier champ (exclus) sur lequel va porter le tri (ici le champ n°2):


```
$ sort -t: +1 -2 clients
M:ABADIE:Marcel:42:Lannemezan
M:DUPOND:Gerard:38:Toulouse
F:DURAND:Claude:25:Auch
M:DURAND:Bernard:33:Tournay
F:DURANT:Sophie:24:Albi
M:ESCLOP:Auguste:17:Aubin
$
```

● L'option -r permet de trier en ordre inverse:

```
$ sort -t: +1 -2 -r clients
M:ESCLOP:Auguste:17:Aubin
F:DURANT:Sophie:24:Albi
F:DURAND:Claude:25:Auch
M:DURAND:Bernard:33:Tournay
M:DUPOND:Gerard:38:Toulouse
M:ABADIE:Marcel:42:Lannemezan
$
```

● L'option -n permet d'effectuer un tri numérique:

```
$ sort -t: +3 -n clients
M:ESCLOP:Auguste:17:Aubin
F:DURANT:Sophie:24:Albi
F:DURAND:Claude:25:Auch
M:DURAND:Bernard:33:Tournay
M:DUPOND:Gerard:38:Toulouse
M:ABADIE:Marcel:42:Lannemezan
$
```

La commande sort

Définition

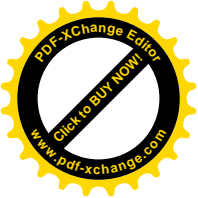
Cette commande permet de trier des fichiers. Les lignes sont classées en fonction d'un critère déterminé selon un N° de champ dans la ligne.

Syntaxe

La syntaxe de base est :

```
sort [-t caractère +pos1 [-pos2]] ] [-r] [-n] [-o fichier-sortie] [-f]  
fichier1 [fichier2 ...]
```

Exercice



- Créez un fichier `client` avec le contenu suivant :

```
Regis;Etain;1000.00
Ursula;Forget;500.00
Hugues;Jonas;250.00
Daniel;Martin;600.00
Marthe;Pommier;4500.00
```

les valeurs numériques correspondent, par exemple, au crédit maximum de chaque client.

- Triez les clients par ordre ascendant de crédit.
- Triez les clients par ordre descendant de crédit.
- Triez les clients par leur nom dans l'ordre alphabétique inverse.
- Triez les clients par leur prénom dans l'ordre alphabétique.

COMPTER LES CARACTERES, MOTS, LIGNES D'UN FICHIER



- Voici le contenu du fichier `texte` :

```
$ cat texte
Voici mon fichier. Un petit
texte pour la commande wc.
Une ligne supplémentaire, pour
ce fichier contenant 4 lignes.
$
```

- `wc` affiche le nombre de lignes, mots et caractères du fichier `texte`:

```
$ wc texte
4 19 118 texte
$
```

- L'option `-l` n'affiche que le nombre de lignes :

```
$ wc -l texte
4 texte
$
```

- L'option `-c` n'affiche que le nombre de caractères:

```
$ wc -c texte
118 texte
$
```

● L'option `-cw` affiche le nombre de caractères et de mots:

```
$ wc -cw texte
118 19 texte
$
```

La commande `wc`

Définition

La commande `wc` compte les mots, les lignes et/ou caractères d'un fichier.

Syntaxe

```
wc [-lwc] fichier
```

Exercice

- Créez un fichier texte quelconque
- Comptez le nombre de mots du fichier créé
- Comptez le nombre de lignes
- Testez la commande `wc` sans option

AFFICHER LES NUMEROS DE LIGNES D'UN FICHIER



● Voici le contenu du fichier `texte` :

```
$ cat texte
Voici mon fichier. Un petit
texte pour la commande wc.
Une ligne supplémentaire, pour
ce fichier contenant 4 lignes.
$
```

● `nl` affiche le fichier en faisant précéder chaque ligne par son n°:

```
$ nl texte
1  Voici mon fichier. Un petit
2  texte pour la commande wc.
3  Une ligne supplémentaire, pour
4  ce fichier contenant 4 lignes.
```

La commande nl

Définition

La commande `nl` numérote les lignes d'un fichier.

Syntaxe

`nl fichier`

Exercice

- affichez le fichier `/etc/passwd` avec n° de lignes.

MANIPULATION DE REPERTOIRES



Les répertoires servent à ranger des fichiers et/ou catalogues.

L'espace de travail de l'utilisateur est une arborescence de répertoires et de fichiers.

Des commandes UNIX permettent d'organiser et de gérer cette hiérarchie (créer, effacer, parcourir, ...)

Il y a un catalogue particulier à chaque usager: ***le catalogue personnel***.

Ce catalogue est le sommet de l'arborescence de l'espace de travail de l'utilisateur, c'est le catalogue dans lequel il est placé à la connexion (`home directory`).

Le catalogue de connexion est repéré par la variable d'environnement `HOME`, mais aussi par le caractère `~`.

Généralement le nom de votre répertoire personnel est identique à votre [nom d'utilisateur](#).

Les principales commandes sur les répertoires:

- Afficher le répertoire courant : `pwd`
- Se déplacer dans l'arborescence : `cd`
- Créer un répertoire : `mkdir`
- Effacer un répertoire : `rmdir`
- Copier un répertoire : `cp`

AFFICHER LE REPERTOIRE COURANT

Le [répertoire courant](#) est le catalogue dans lequel vous êtes en train de travailler. Initialement le répertoire courant est le catalogue de connexion.

Il est nécessaire de connaître sa position dans l'arborescence du système à tout instant.

```
$ pwd
/users/fudmip/ubanel1/cours
$
```

Dans l'exemple ci-dessus la commande `pwd` nous indique que nous sommes dans le répertoire `cours` placé dans `ubanel1` lui-même placé dans `fudmip` du catalogue `users` situé sous la racine.



La commande `pwd`



Définition

La commande `pwd` affiche à l'écran le chemin d'accès au catalogue courant.



Syntaxe

`pwd`

Remarque

Certains systèmes maintiennent une variable `PWD` qui contient le chemin d'accès au catalogue courant.

En [shell](#) `csh` (`tcsh`, ...) la variable `cwd` contient aussi le catalogue courant.



Exercice

- Activez la commande `pwd`
- Tapez la commande:

```
echo $SHELL ":" $PWD ":" $cwd
```

SE DEPLACER DANS UN REPERTOIRE

● La commande `cd` sans argument permet de revenir dans le [home directory](#).

```
$ pwd
/etc
$ echo $HOME
/users/fudmip/ubane11
$ cd
$ pwd
/users/fudmip/ubane11
$
```

● La commande `cd` permet d'aller dans un répertoire (dont on donne le chemin d'accès ici en *absolu*).

```
$ pwd
/users/fudmip/ubane11/cours
$ cd /usr/bin
$ pwd
/usr/bin
$
```

● En spécifiant un chemin d'accès *relatif*.

```
$ pwd
/users/fudmip/ubane11/cours
$ cd boite
$ pwd
/users/fudmip/ubane11/cours/boite
$
```

■ Pour remonter dans la hiérarchie il suffit d'utiliser la convention ..:

```
$ pwd
/users/fudmip/ubane11/cours/boite
$ cd ..
$ pwd
/users/fudmip/ubane11/cours
$
```

■ Il est possible de remonter plusieurs fois ... pour éventuellement redescendre:

```
$ pwd
/users/fudmip/ubane11/cours/boite
$ cd ../../utilitaire/cci
$ pwd
/users/fudmip/ubane11/utilitaire/cci
$
```

La commande cd

Définition

Cette commande permet de se déplacer dans l'arborescence des catalogues existants sur le système.

Syntaxe

```
cd
cd [ chemin relatif ]
cd [ chemin absolu ]
```

Exercice

- Affichez votre répertoire courant.
- Placez-vous dans `/tmp`.
- Affichez votre répertoire courant.
- Allez dans le catalogue `lib` qui se trouve sous `/usr` en spécifiant un chemin d'accès **relatif**.
- Affichez votre répertoire courant.
- Revenez dans votre répertoire de connexion.

CREER UN REPERTOIRE



La commande `mkdir` crée le répertoire passé en paramètre:

```
$ ls -l
total 6
-rw-r--r--  1 ubanell  fudmip      152 janv 15 11:22 Fic4
-rw-r--r--  1 ubanell  fudmip      152 janv 15 11:22 Fich
drwxrwxr-x  2 ubanell  fudmip    1024 janv 15 11:23 Katal
$ mkdir rapport
$ ls -l
total 8
-rw-r--r--  1 ubanell  fudmip      152 janv 15 11:22 Fic4
-rw-r--r--  1 ubanell  fudmip      152 janv 15 11:22 Fich
drwxrwxr-x  2 ubanell  fudmip    1024 janv 15 11:23 Katal
drwxrwxr-x  2 ubanell  fudmip      24 janv 17 11:13 rapport
$ mkdir ../programme
$ ls -l ..
total 12
drwxrwxr-x  4 ubanell  fudmip    1024 janv 17 11:13 boite
-rw-r--r--  1 ubanell  fudmip      160 janv 15 18:27 clients
-rw-r--r--  1 ubanell  fudmip      391 janv 12 11:29 Fich.c
-rw-rw-r--  1 ubanell  fudmip       33 janv 15 16:35 ll
drwxrwxr-x  2 ubanell  fudmip      24 janv 17 11:14 programme
drwxrwxr-x  2 ubanell  fudmip    1024 janv 17 11:09 recup
$ mkdir /tmp/essai
$ ls -l /tmp
total 4
drwxrwxr-x  2 ubanell  fudmip      24 janv 17 11:14 essai
drwxrwxrwx  2 baque    fudmip      24 janv 17 11:09 ii
$
```

La commande `mkdir`



Définition

Cette commande permet de créer des répertoires, il faut bien sûr pouvoir le faire, c'est-à-dire être dans son espace de travail.

Syntaxe

`mkdir [-p] nom-répertoire`



Exercice :

- Créer le catalogue `Ktal` dans votre *home directory*.
- Créer le catalogue `/tmp/$USER.Ktal/Ktal` en une seule commande `mkdir`.

DETRUIRE UN REPERTOIRE

La commande `rmdir` détruit le catalogue passé en paramètre:

```
$ ls -l
total 6
-rw-r--r--  1 ubanell  fudmip      391 janv 12 11:29 Fich.c
drwxrwxr-x  2 ubanell  fudmip      24 janv 17 11:43 prog-C
drwxrwxr-x  2 ubanell  fudmip    1024 janv 17 11:38 prog-PERL
$ rmdir prog-C
$ ls -l
total 4
-rw-r--r--  1 ubanell  fudmip      391 janv 12 11:29 Fich.c
drwxrwxr-x  2 ubanell  fudmip    1024 janv 17 11:38 prog-PERL
$
```

Si le catalogue n'est pas vide `rmdir` refuse de le détruire:

```
$ ls -lR
total 4
-rw-r--r--  1 ubanell  fudmip      391 janv 12 11:29 Fich.c
drwxrwxr-x  2 ubanell  fudmip    1024 janv 17 11:38 prog-PERL

./prog-PERL:
total 6
-rw-rw-r--  1 ubanell  fudmip        0 janv 17 11:38 Fic.old
-rw-rw-r--  1 ubanell  fudmip      44 janv 17 11:38 ii
-rw-rw-r--  1 ubanell  fudmip      62 janv 17 11:38 jj
-rw-rw-r--  1 ubanell  fudmip      33 janv 17 11:38 ll
$ rmdir prog-PERL
rmdir: prog-PERL: Répertoire qui n'est pas vide
$
```

● Pour effacer un catalogue non-vidé, il suffit de le vider avant.

La commande `rm prog-PERL/*` permet d'effacer tous les fichiers contenus dans le répertoire `prog-PERL`:

```
$ ls -lR
total 4
-rw-r--r--  1 ubane11  fudmip      391 janv 12 11:29 Fich.c
drwxrwxr-x  2 ubane11  fudmip    1024 janv 17 11:38 prog-PERL

./prog-PERL:
total 6
-rw-rw-r--  1 ubane11  fudmip        0 janv 17 11:38 Fic.old
-rw-rw-r--  1 ubane11  fudmip      44 janv 17 11:38 ii
-rw-rw-r--  1 ubane11  fudmip      62 janv 17 11:38 jj
-rw-rw-r--  1 ubane11  fudmip      33 janv 17 11:38 ll

$ rm prog-PERL/*
$ ls -lR
total 4
-rw-r--r--  1 ubane11  fudmip      391 janv 12 11:29 Fich.c
drwxrwxr-x  2 ubane11  fudmip    1024 janv 17 11:46 prog-PERL

./prog-PERL:
total 0

$ rmdir prog-PERL
$ ls -l
total 2
-rw-r--r--  1 ubane11  fudmip      391 janv 12 11:29 Fich.c
$
```

● Il est toutefois possible d'effacer un catalogue non-vidé en utilisant la commande `rm` avec l'option `-r` (à utiliser avec précaution):

```
$ ls -lR
total 4
-rw-r--r--  1 ubane11  fudmip      391 janv 12 11:29 Fich.c
drwxrwxr-x  2 ubane11  fudmip    1024 janv 17 11:47 prog-PERL

./prog-PERL:
total 6
-rw-rw-r--  1 ubane11  fudmip        0 janv 17 11:47 Fic.old
-rw-rw-r--  1 ubane11  fudmip      44 janv 17 11:47 ii
-rw-rw-r--  1 ubane11  fudmip      62 janv 17 11:47 jj
-rw-rw-r--  1 ubane11  fudmip      33 janv 17 11:47 ll

$ rm -rf prog-PERL
$ ls -lR
total 2
-rw-r--r--  1 ubane11  fudmip      391 janv 12 11:29 Fich.c
$
```

La commande rmdir

Définition

La commande `rmdir` permet de détruire des catalogues vides.

Syntaxe

```
rmdir [-f | -i] [-p] répertoire
rm -R   répertoire
```

Exercice :

- Effacer le catalogue précédemment créé sous `/tmp (/tmp/$USER.Ktal/Ktal vu dans l'exercice de mkdir).`

COPIER UN REPERTOIRE

● L'option `-R` de la commande `cp` permet de copier des catalogues:

```
$ ls -lR prog-PERL/
total 6
-rw-rw-r-- 1 ubane11 fudmip      44 janv 17 11:45 ii
-rw-rw-r-- 1 ubane11 fudmip      62 janv 17 11:45 jj
-rw-rw-r-- 1 ubane11 fudmip      33 janv 17 11:45 ll
$ cp -R prog-PERL prog-sauve
$ ls -lR prog-sauve
total 6
-rw-rw-r-- 1 ubane11 fudmip      44 janv 17 12:27 ii
-rw-rw-r-- 1 ubane11 fudmip      62 janv 17 12:27 jj
-rw-rw-r-- 1 ubane11 fudmip      33 janv 17 12:27 ll
$
```

La commande cp avec option -r



Définition

Il est possible de dupliquer le contenu d'un répertoire en utilisant la commande **cp** (copy) et l'option **-r** (récursive).

De cette façon, tous les fichiers contenus dans tous les sous-répertoires du répertoire copié seront copiés également.



Syntaxe

cp -r *répertoire-a-copier* *nouveau-répertoire*



Exercice :

- Créez un premier répertoire que vous appellerez *source* dans votre répertoire personnel en tapant la commande **mkdir source**
- Placez vous dans ce répertoire **cd source** et créez des fichiers textes quelconques à l'aide de la commande **cat** , par exemple **cat fichier1**, **cat fichier2**, **cat fichier3**, etc
...
- Maintenant, remontez dans le répertoire parent en tapant la commande **cd ..**
- Dupliquez le répertoire *source* dans le répertoire *cible* en tapant la commande **cp -r source cible**
- Assurez vous que les fichiers contenus dans le répertoire *source* existent bien dans le répertoire *cible* en tapant :

cd cible

ls

ou en tapant tout simplement

ls cible



UNIX

Chapitre 3 : Notion de processus dans le système UNIX

NOTION DE PROCESSUS



Un processus est un programme en cours d'exécution

Un **programme**, produit par un éditeur de liens, est un fichier binaire exécutable mémorisé sur disque.

Pour l'exécuter le système le charge en mémoire, il devient alors un processus .

Un processus est identifié au sein du système par un n° unique, le PID .

Les commandes de gestion de processus:

- Etat des processus actifs : ps
- Arrêter un processus actif : kill
- Différentes façons de lancer un processus (nohup , at time, batch , nice).

AFFICHER L'ETAT DES PROCESSUS ACTIFS

La commande `ps` affiche la liste des processus actifs :

```
$ ps
  PID TTY          TIME COMMAND
 11223 ttyp4        0:00 tcsh
 12609 ttyp4        0:00 ps
 11224 ttyp4        0:02 xv
$
```

ne sont affichés que les processus lancés dans la fenêtre terminal courante.

L'option `-u` permet d'afficher tous les processus appartenant à l'utilisateur spécifié:

```
$ ps -u $USER
  PID TTY          TIME COMMAND
 11222 ?            0:00 hpterm
 11597 ?            0:00 hpterm
 11614 ?            0:00 vuepad
 11126 ?            0:00 hpterm
 11219 tttyp1        0:00 vuepad
10995 ?            0:00 vuesession
 11223 tttyp4        0:00 tcsh
 12610 tttyp4        0:00 ps
 11123 tttyp1        0:00 softmsgsrv
 11598 tttyp5        0:00 tcsh
 11127 tttyp2        0:00 tcsh
 11119 ?            0:13 vuewm
 11128 tttyp2        1:08 netscape
 11220 tttyp2        0:43 xemacs-19.13
 11224 tttyp4        0:02 xv
$
```

sont affichés tous les processus appartenant à l'utilisateur pour la session courante.

La commande ps

Définition

Cette commande affiche la liste des processus actifs sur le système.
Il existe 2 types de processus : les processus systèmes qui accomplissent des services généraux et les processus utilisateurs.
Par défaut, la commande `ps` n'affiche que les processus utilisateurs.

Syntaxe

```
ps [-u utilisateur] [-e] [-f]
```

■ Les options `-ef` affichent les informations complètes sur tous les processus:

<u>UID</u>	<u>PID</u>	<u>PPID</u>	<u>c</u>	<u>STIME</u>	<u>TTY</u>	<u>TIME</u>	<u>COMMAND</u>
root	0	0 0		Jan 1	?	0:06	swapper
root	1	0 0		Jan 9	?	0:00	init
root	2	0 0		Jan 9	?	0:01	vhand
root	3	0 0		Jan 9	?	0:00	statdaemon
root	7	0 0		Jan 9	?	0:00	unhashdaemon



```
root      10      0 0   Jan 9  ?      0:00 syncdaemon
root     417      1 0   Jan 9  console 0:00 /etc/getty -h console console
root     418      1 0   Jan 9  ?      0:00 /usr/vue/bin/vuelogin
root     169      1 0   Jan 9  ?      0:00 /etc/inetd
root     239      1 0   Jan 9  ?      0:00 /usr/local/bin/httpd -d
          /usr/local/httpd
root     237      1 0   Jan 9  ?      1:26 /usr/bin/X11/fs -daemon
ubane11 28693 28417 0 14:01:11 tttyp3 6:49 xemacs
tukalo   3755 28255 0 15:44:42 ttyv2  1:20 /usr/vue/bin/vuepad
tukalo  28255 28254 0 13:47:41 ttyv2  0:00 -tcsh
daemon   419    418 1   Jan 9  ?      79:44 /usr/bin/X11/X :0
root    22060    169 0 09:38:36 ttyv0  0:00 telnetd
ubane11  8180 28593 7 17:20:28 tttyp4  0:00 ps -ef
ubane11 28416 28409 0 13:51:41 ?      0:07 /usr/vue/bin/hpterm
ubane11  7964  7961 0 17:00:38 tttyp5  0:00 tcsh
ubane11 28594 28593 0 13:58:31 tttyp4  0:03 xv
ubane11 28417 28416 0 13:51:41 tttyp3  0:00 tcsh
ubane11   913 28409 0 14:51:45 ?      0:16 /usr/audio/bin/audio_editor
baque    7138      1 0   Jan 15 ?      34:34 xemacs
ubane11  8097 28409 0 17:11:01 ?      0:00 /usr/vue/bin/vuepad
baque   25516 22061 0 10:28:14 ttyv0  0:00 tcsh
root    28277    418 0 13:50:59 ?      0:00 /usr/vue/bin/vuelogin
ubane11 28593 28592 0 13:58:28 tttyp4  0:00 tcsh
root    28254    169 0 13:47:41 ttyv2  0:00 telnetd
ubane11 28592 28409 0 13:58:28 ?      0:02 /usr/vue/bin/hpterm
ubane11 28409 28285 0 13:51:27 ?      0:27 vuewm
ubane11 28285 28277 0 13:51:17 ?      0:00 /usr/vue/bin/vuesession
baque   22061 22060 0 09:38:36 ttyv0  0:00 -tcsh
ubane11  7961 28409 0 17:00:36 ?      0:00 /usr/vue/bin/hpterm
```

ARRETER UN PROCESSUS ACTIF

🔵 La commande `kill` arrête un processus dont on fournit le numéro (PID) :

```
$ ps
  PID TTY          TIME COMMAND
   777 tttyp2        0:00 tcsh
   943 tttyp2        0:00 xv
   964 tttyp2        0:00 ps
$ ps
  PID TTY          TIME COMMAND
   777 tttyp2        0:00 tcsh
   943 tttyp2        0:00 xv
   980 tttyp2        0:00 ps
$ kill 943
[1]      Terminated                  xv
$ ps
  PID TTY          TIME COMMAND
   777 tttyp2        0:00 tcsh
  1026 tttyp2        0:00 ps
$
```

● L'option `-9` permet de forcer la destruction d'un processus :

```
$ ps
  PID TTY          TIME COMMAND
  2571 tttyp2        0:00 vuepad
   777 tttyp2        0:00 tcsh
  2686 tttyp2        0:00 ps
$ kill -9 2571
[1]      Killed                        /usr/vue/bin/vuepad
$ ps
  PID TTY          TIME COMMAND
  2767 tttyp2        0:00 ps
   777 tttyp2        0:00 tcsh
$
```

La commande kill

Définition

Cette commande interrompt un processus en cours d'exécution.
En réalité `kill` envoie un signal au processus spécifié.

Syntaxe

```
kill [-signal] PID
```




Exercice :

- Lancez un éditeur de texte.
- Identifiez son numéro de processus avec la commande `ps`.
- Tuez cet éditeur à l'aide de la commande `kill`.

DIFFERENTES FAÇONS DE LANCER UN PROCESSUS

Laisser vivre un processus même après la fin d'une session (logout) [nohup](#)

Lancer un processus en différé [at time, batch](#)

Lancer un processus avec une priorité particulière [nice](#)

Continuité d'exécution d'un processus : nohup



Définition

Lors de la déconnexion, les processus lancés par l'utilisateur qui s'exécutent encore sont tués automatiquement. Toutefois il est possible de prolonger l'exécution d'un processus en utilisant cette commande.



Syntaxe

```
nohup commande [parametres]
```

Exécution d'un processus en différé : at time, batch



Définition

at : permet de spécifier le moment de l'exécution.

batch : la commande est mise en attente et est exécutée quand le système n'est pas surchargé



Syntaxe

```
at time commande [parametres]
```

```
batch commande [parametres]
```

■ Exemple :

```
$ at 8:15 am Jan 23 mon-prog
```

Exécution d'un processus avec priorité basse : nice



Définition

La commande `nice` permet de donner des priorités plus ou moins élevées pour l'exécution de processus selon l'importance de la tâche qu'ils remplissent.



Syntaxe

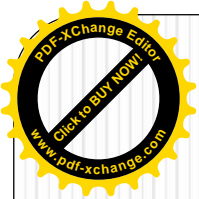
```
nice [+/-nombre] commande [parametres]
```

avec 1 (fort) \leq nombre \leq 19 (faible).

Par défaut, nombre est égal à 10.

■ Remarque :

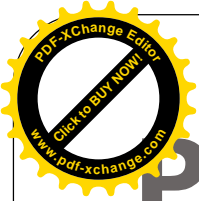
Seul le SU (Super Utilisateur) peut exécuter un processus en augmentant sa priorité. Un utilisateur normal ne peut que la diminuer.



RESEAUX ET PROTOCOLES

CHAPITRE 3 : PROTOCOLE IP

Le contenu de ce document est soumis à la Licence de Documentation Libre (**GNU Free Documentation License**).



Plan du Chapitre

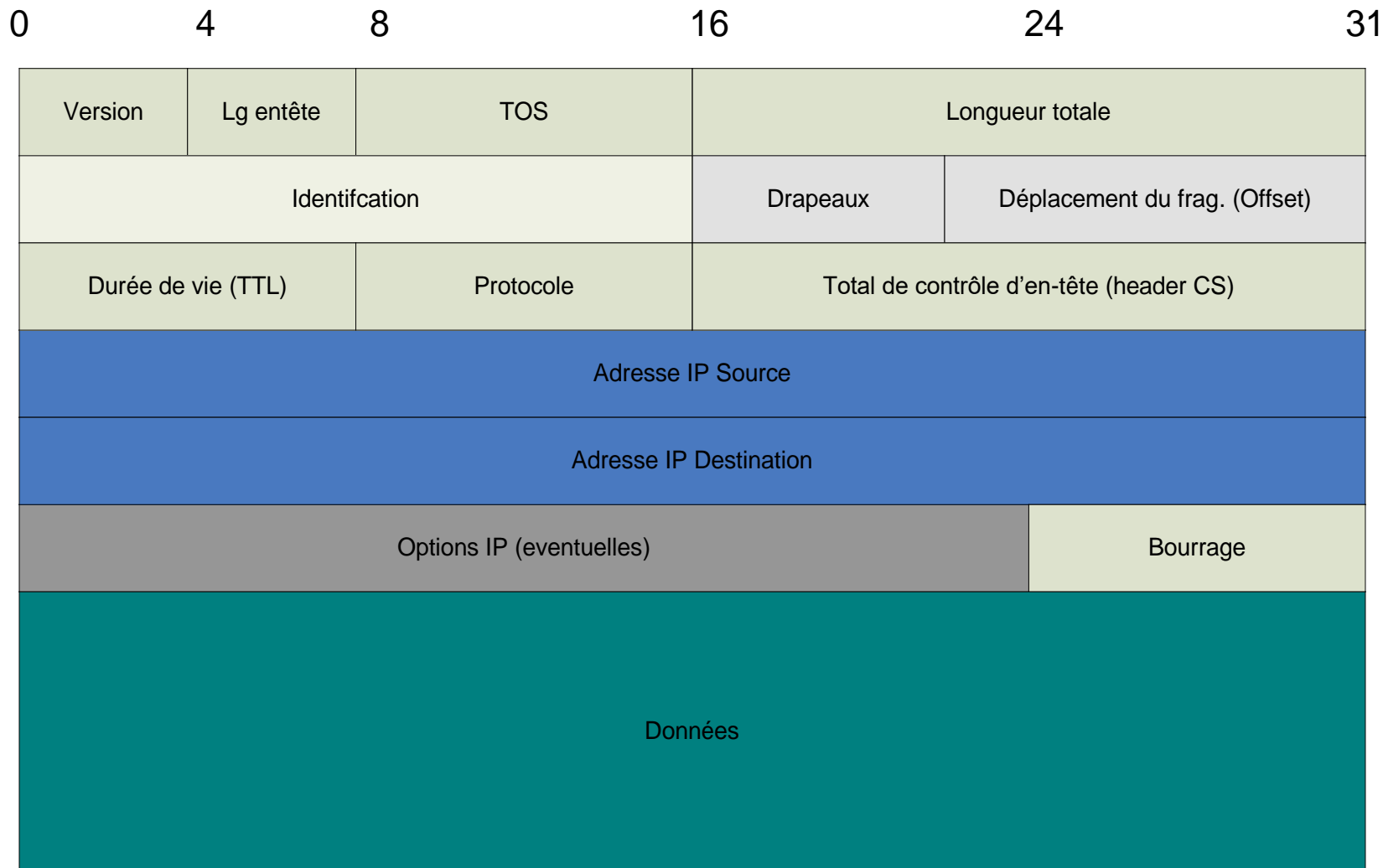
1. DATAGRAMME IP
2. FORMAT DE DATAGRAMME
3. STRUTURE D' ADRESSE IP
4. MASQUE DE SOUS-RESEAU

Datagramme IP

- **Qu'est ce qu'un datagramme**

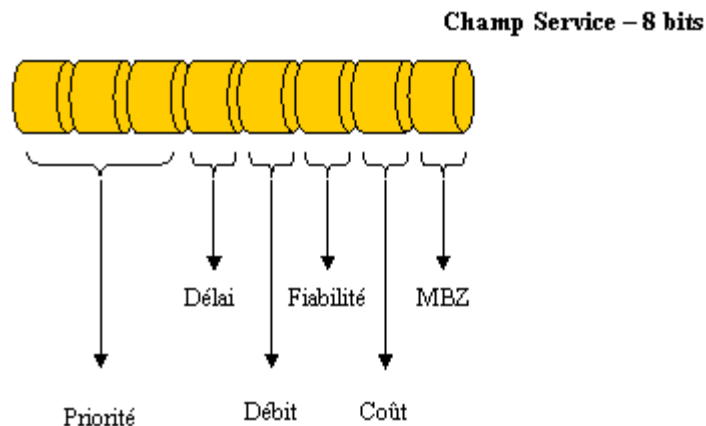
- Le protocole Internet (IP) est la méthode d'adressage favorisée des réseaux hiérarchiques. Le protocole IP est le protocole réseau d'Internet. À mesure que les données circulent vers le bas du modèle OSI, elles sont encapsulées au niveau de chaque couche. Au niveau de la couche réseau, les données sont encapsulées dans des paquets (aussi appelés datagrammes).
- Le protocole IP détermine le format de l'en-tête IP (qui comprend les informations d'adressage et de contrôle), mais ne se préoccupe pas des données proprement dites.

Structure d'un datagramme IP



Explication des champs de Datagramme (1,

- *Version* - indique la version de protocole IP utilisée (4 bits). Exemple 04 – IPV4, 05 – ST Datagram Mode , 06 – IPV6
- *HLEN (IP header length - Longueur de l'en-tête IP)* - indique la longueur de l'en-tête du datagramme en mots de 32 bits (4 bits). $5 \times 4 = 20 \text{ octets}$. Ce champ permet de savoir où commencent les données du paquet.
- *Type de service* - indique l'importance qui lui a été accordée par un protocole de couche supérieure donné (8 bits).



- Le champ *Priorité* est codé sur 3 bits. Il indique la priorité que possède la paquet. Voici les correspondances des différentes combinaisons :
 - 0 – 000 – Routine
 - 1 – 001 – Prioritaire
 - 2 – 010 – Immédiat
 - 3 – 011 – Urgent
 - 4 – 100 – Très urgent
 - 5 – 101 – Critique
 - 6 – 110 – Supervision interconnexion
 - 7 – 111 – Supervision réseau

Explication des champs de Datagramme (2)

- *Longueur totale* - Le champ Longueur totale est codé sur 16 bits et représente la longueur du paquet incluant l'entête IP et les Data associées. La longueur totale est exprimée en octets, ceci permettant de spécifier une taille maximum de $2^{16} = 65535$ octets. La longueur des Data est obtenu par la combinaison des champs IHL et Longueur totale :
 - $Longueur_des_data = Longueur_totale - (IHL * 4);$
- *Identification* — Le champ identification permet d'identifier le paquet. Lorsqu'un paquet a été fragmenté, le récepteur utilise le numéro d'identification pour déterminer quels fragments font partie du paquet à réassembler.

Explication des champs de Datagramme (3,

- **Flags** (3 bits) :
 - 1 bit réservé dont la valeur est égale à 0.
 - **DF = Don't Fragment** (1 bit) : indique si le paquet peut être fragmenté (0) ou pas (1). Si un paquet ne pouvant pas être fragmenté arrive sur un routeur mais est trop grand pour être transféré (taille du paquet supérieure à la MTU du port de sortie), le paquet sera détruit.
 - **MF = More Fragments** (1 bit) : indique s'il y a d'autres fragments (1) ou si celui-ci est le dernier (0). Si ce bit est à 0, cela peut aussi vouloir dire que le paquet n'a pas été fragmenté (dans ce cas, le champ **Fragment Offset** sera aussi à 0).

Explication des champs de Datagramme (4, 5)

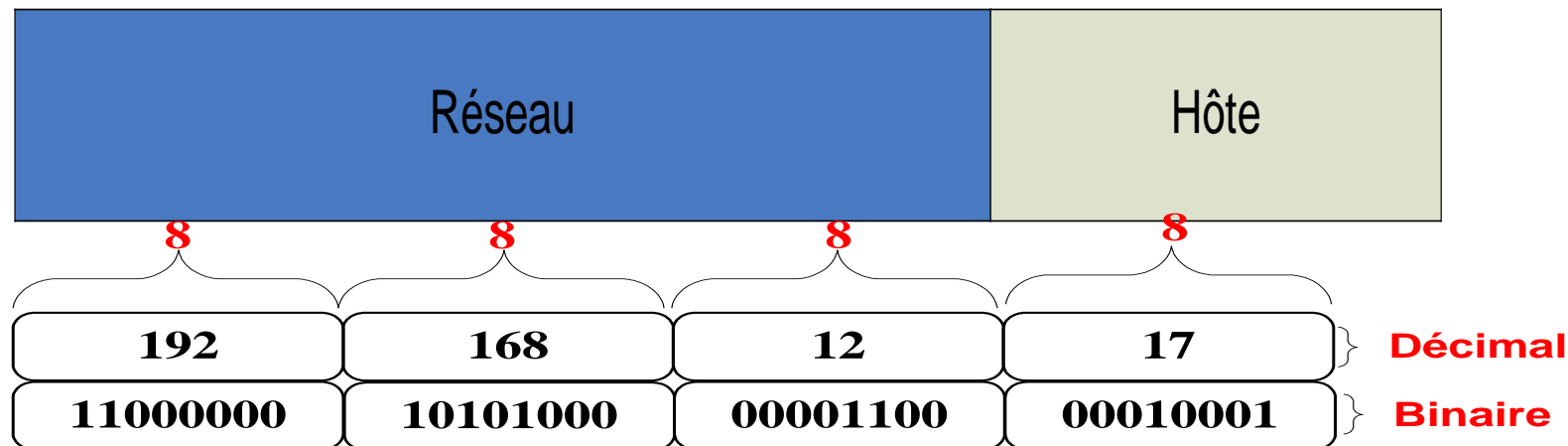
- **Fragment Offset** (13 bits) : Le champ Position fragment est codé sur 13 bits et indique la position du fragment par rapport à la première trame. Le premier fragment possède donc le champ Position fragment à 0.
- **Time To Live** (8 bits) : sa valeur est initialisée par l'expéditeur et est décrétementée à chaque fois que le paquet franchit un routeur. Quand le **Time To Live** (ou **TTL**) atteint 0, le paquet est détruit.
- **Protocol** (8 bits) : indique le protocole encapsulé dans les données du paquet. Des valeurs courantes de ce champ sont 1 (ICMP), 2 (IGMP), 6 (TCP) ou 17 (UDP).

Explication des champs de Datagramme (5)

- **Header Checksum** (16 bits) : utilisé pour vérifier que l'en-tête IPv4 n'a pas été altéré durant la transmission. Lorsqu'un routeur reçoit le paquet, il recalcule la somme de contrôle et la compare avec la valeur de ce champ. S'il y a une différence, le paquet est détruit. Le **Header Checksum** est recalculé et mis à jour à chaque routeur traversé (puisque au minimum le **TTL** a diminué).
- **Source Address** (32 bits) : adresse IPv4 de l'expéditeur du paquet.
- **Destination Address** (32 bits) : adresse IPv4 du destinataire du paquet.
- **Options** (0 - 320 bits) : ce champ facultatif permet d'ajouter des options à l'en-tête IPv4 (routage défini par la source, enregistrement de route, ...).
- **Bourrage** (0 - 31 bits) : si nécessaire, des bits à 0 sont ajoutés aux options pour que la longueur de l'en-tête soit toujours un multiple de 32 bits (en effet, le champ **IHL** compte en mots de 32 bits).

Format d'adressage IP

- Une adresse IP est codée sur 32 bits. Elle comprend deux parties principales, un numéro de réseau et un numéro de machine.
- Le numéro de réseau d'une adresse IP identifie le réseau auquel une unité est connectée, alors que la partie hôte d'une adresse IP pointe vers une unité spécifique de ce réseau.
- Comme il est pratiquement impossible de mémoriser 32 bits, les adresses IP sont divisées en groupes de 8 bits séparés par des points, et représentées dans un format décimal et non binaire.



Différentes classes d'adressage IP

- Un organisme peut recevoir trois classes d'adresses IP de l'InterNIC (Internet Network Information Center) (ou de son fournisseur de services Internet). Il s'agit des classes A, B et C. L'InterNIC réserve à présent les adresses de classe A aux gouvernements du monde entier et les adresses de classe B aux entreprises de taille moyenne. Tous les autres demandeurs reçoivent des adresses de classe C.

- **La classe A**

- Toutes les adresses IP de classe A n'utilisent que les huit premiers bits pour indiquer la partie réseau de l'adresse. Les trois octets restants peuvent servir pour la partie hôte de l'adresse.

- **La classe B**

- Toutes les adresses IP de classe B utilisent les 16 premiers bits pour indiquer la partie réseau de l'adresse. Les deux octets restants de l'adresse IP sont réservés à la partie hôte de l'adresse.

- **La classe C**

- Toutes les adresses IP de classe C utilisent les 24 premiers bits pour indiquer la partie réseau de l'adresse. Seul le dernier octet d'une adresse IP de classe C est réservé à la portion hôte de l'adresse.

- **La classe D**

- Classe D est une classe réservée pour la diffusion multicast

- **La classe E**

- Classe E est une classe expérimentale qui est utilisée pour la recherche.

Plages d'adressage IP

Classes	Bits de poids fort	Plages	Masques par défaut
A	0 xxxxxxx	0 à 127	255.0.0.0
B	10 xxxxxx	128 à 191	255.255.0.0
C	110 xxxxx	192 à 223	255.255.255.0
D	1110 xxxx	224 à 239	Aucun
E	1111 xxxx	240 à 255	Aucun

Plages d'adresses privées :

10.0.0.0 à 10.255.255.255

172.16.0.0 à 172.31.255.255

192.168.0.0 à 192.168.255.255

Plages d'adresses réservées :

0.0.0.0 à 0.255.255.255 (interdit)

127.0.0.0 à 127.255.255.255 (Loopback)

224.0.0.0 à 239.255.255.255 (Multicast)

255.255.255.255 (Broadcast)

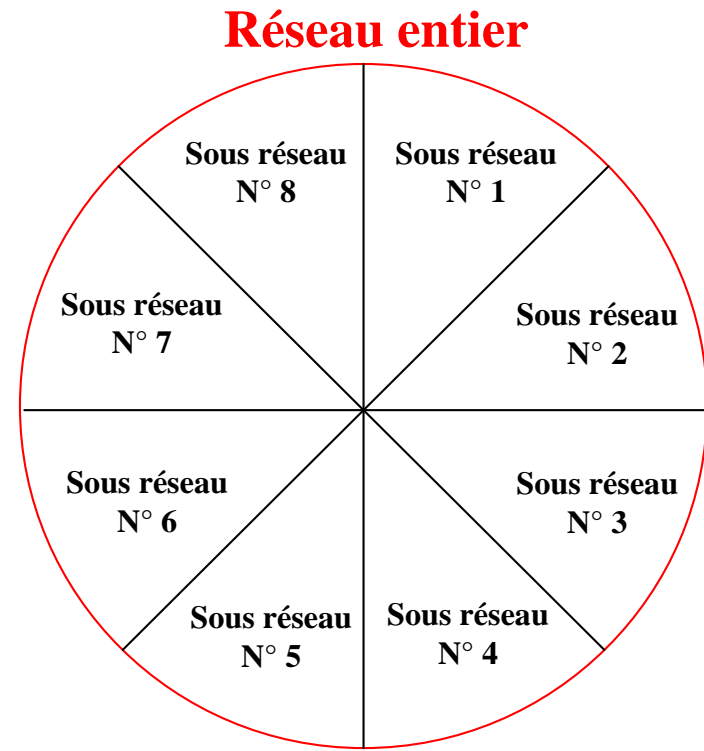


Masque sous-réseau

- Qu'est-ce que le masque de sous-réseau ?
 - subdivision d'un réseau en plusieurs sous réseaux
- Intérêt
 - Avoir plusieurs de domaines de broadcast plus petit

4.1 Mise en pratique

- Règle pour déterminer le nombre de Sous-Réseaux et d'utilisateurs dans un sous-réseau : $2^n - 2$
- Réseau : 192.168.16.0 /24
- Création de 8 sous-réseaux :
 - 192.168.16.0 /27
 - 192.168.16.32 /27
 - 192.168.16.64 /27
 - 192.168.16.96 /27
 - 192.168.16.128 /27
 - 192.168.16.160 /27
 - 192.168.16.192 /27
 - 192.168.16.224 /27



4.2 Méthode de calcul

- Méthode classique en 6 étapes :
 - 1) Empruntez le nombre de bits suffisants
 - 2) Calculez le nouveau masque de sous réseau
 - 3) Identifiez les différentes plages d'adresses IP
 - 4) Identifiez les plages d'adresses non utilisables
 - 5) Identifiez les adresses de réseau et de broadcast
 - 6) Déterminez les plages d'adresses utilisables par les hôtes

Méthode classique (1/6)

- Énoncé : 150.50.0.0 255.255.0.0 (/16)
- Méthode classique – 1ère étape :
 - Empruntez le nombre de bits suffisants
 - Nombre de machines maximum + 2, par sous-réseau à écrire en binaire et à déterminer le nombre de bits
 - Ex: 500 machines + 2 = 111110110, soit 9 bits empruntés

10010110.00110010.00000000.00000000

7 bits 9 bits

Partie
Sous-
réseau Partie hôte

Méthode classique (2/6)

- Méthode classique – 2^{ème} étape :
- Calculez le nouveau masque de sous réseau

Adresse réseau : **10010110.00110010.00000000.00000000**

7 bits 9 bits

Masque de sous-réseau: **11111111.11111111.11111111.00000000**

Nouveau masque de sous-réseau : **255.255.254.0 (ou /23)**



Méthode classique (3/6)

- Méthode classique – 3^{ème} étape :
- Identifiez les différentes plages d'adresses IP
 - Regarder les bits de la partie sous-réseau
 - Ex: $2^7 - 2 = 126$, soit 126 sous-réseaux utilisables



Méthode classique (3/6)

- Méthode classique – 3^{ème} étape (suite):

7 bits de la partie réseau

0000001|0 = 150.50.2.0
0000010|0 = 150.50.4.0
0000011|0 = 150.50.6.0
0000100|0 = 150.50.8.0
|
|

|
|
|
|
|
|
|

1110111|0 = 150.50.246.0
1111100|0 = 150.50.248.0
1111101|0 = 150.50.250.0
1111110|0 = 150.50.252.0

Méthode classique (4/6)

- Méthode classique – 4^{ème} étape :
- Identifiez les plages d'adresses non utilisables
 - La 1^{ère} plage d'adresse → adresse réseau
 - Ex : 150.50.0.0 /23
 - La dernière plage d'adresse → adresse de broadcast
 - Ex : 150.50.254.0 /23

Méthode classique (5/6)

- Méthode classique – 5^{ème} étape :

- Identifiez les adresses de réseau et de broadcast
 - La 1^{ère} adresse du sous-réseau → adresse de sous-réseau
 - Ex: 150.50.2.0 /23

 - La dernière adresse du sous-réseau → adresse de broadcast
 - Ex : 150.50.3.255 /23

Méthode classique (6/6)

- Méthode classique – 6^{ème} étape :
- Déterminez les plages d'adresses utilisables par les hôtes

<u>Sous-réseau :</u> 150.50.2.0 /23		
150.50.2.0 /23		
150.50.2.1 /23		
150.50.2.2 /23		
	150.50.2.255 /23	
150.50.2.3 /23	150.50.3.0 /23	150.50.3.251 /23
150.50.2.4 /23	150.50.3.1 /23	150.50.3.252 /23
		150.50.3.253 /23
		150.50.3.254 /23
		150.50.3.255 /23

- **Masques de sous-réseaux de longueur variable a été développé pour les raisons suivantes:**
 - **Pour pallier au manque d'adresses IPv4:**
 - Subnetting en 85,
 - Variable lenght subnet Masks (RFC 1009 en 1987),
 - Network address Translation (NAT)
- **Solution ultime: IPv6 (sur 128 bits):**



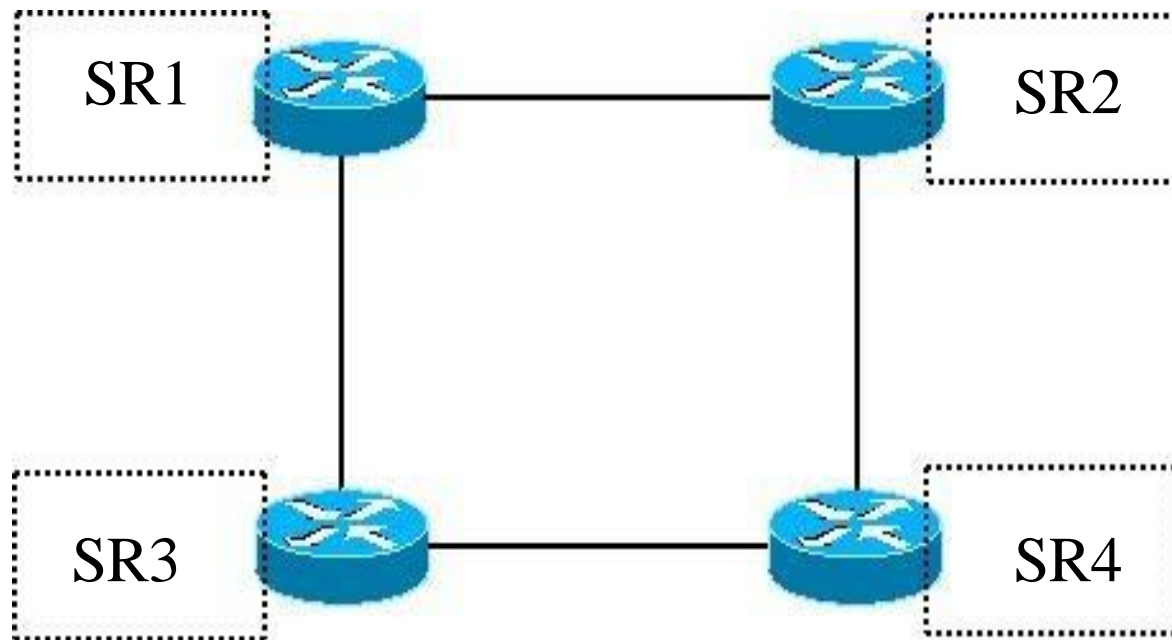
Variable Length Subnet Masks (VLSM)



- **Permet la subdivision d'une adresse de sous-réseaux en sous-réseaux :**
 - Plusieurs masques de sous-réseaux.
 - Obligation d'utiliser un protocole de routage supportant VLSM.
- **Permet de réduire le nombre d'@IP gaspillées dans chaque sous-réseau:**
 - Avec VLSM, certains réseaux peuvent être plus petits et d'autres plus grands, ce qui limite le gaspillage d'adresses IP.

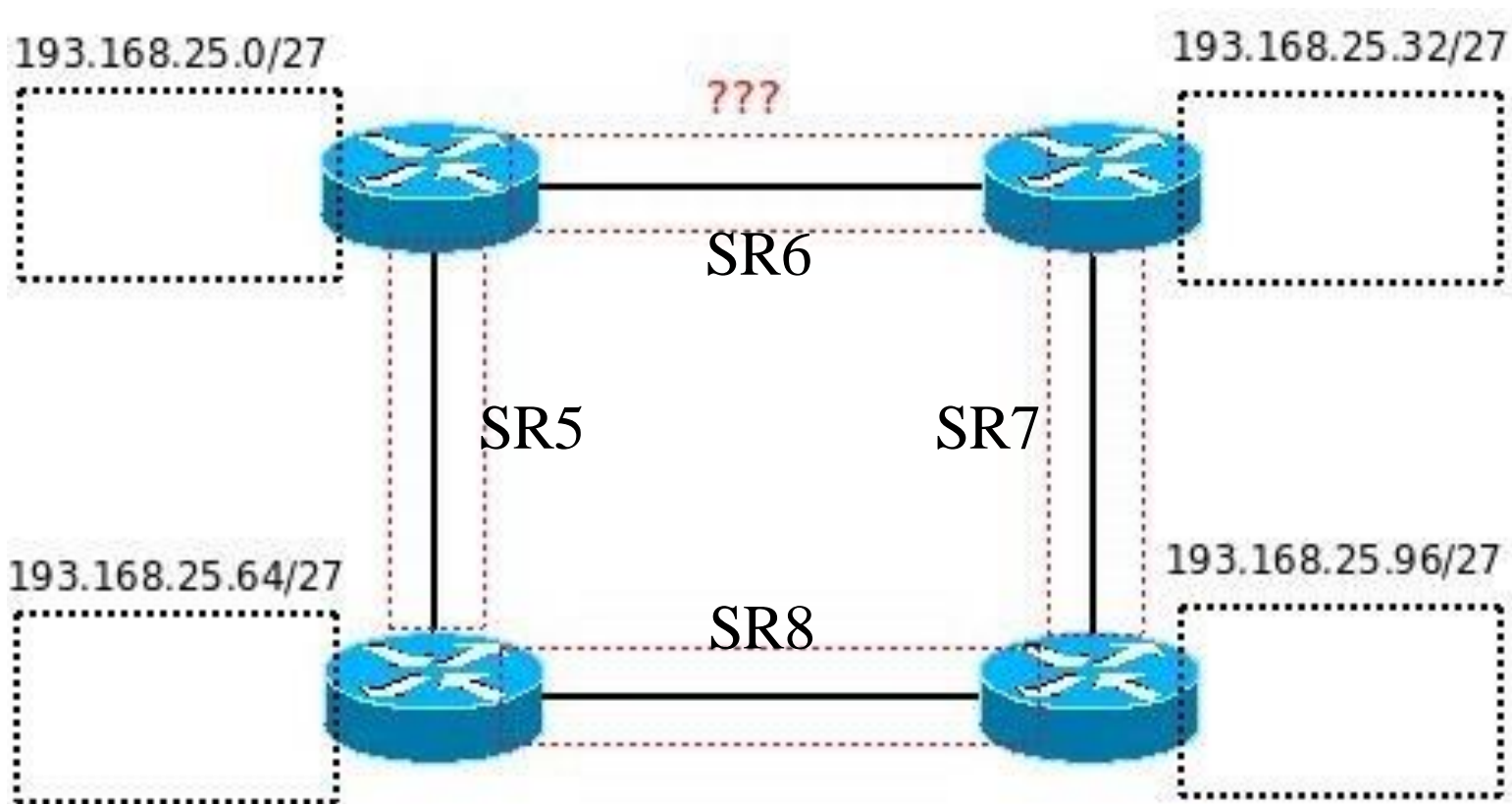
SANS VLSM (1)

- Exemple: Une entreprise ayant une adresse 193.168.25.0/24 veut créer 4 sous-réseaux de 30 machines.



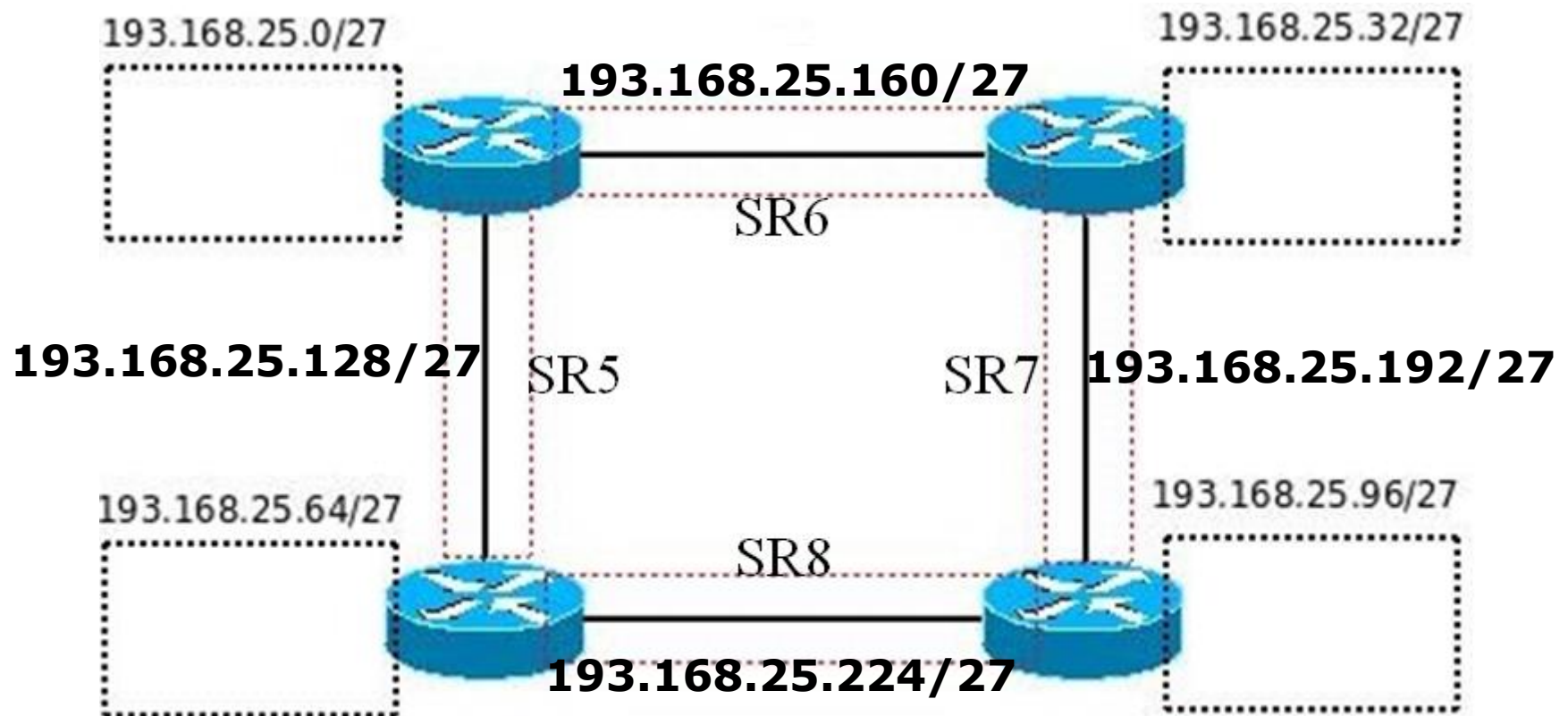
30 machines → **5 bits** id_machine,
3 bits id_sous-réseaux → 8 sous-réseaux disponibles.

SANS VLSM (2)



On a besoin de 4 sous-réseaux supplémentaires.

SANS VLISM (3)

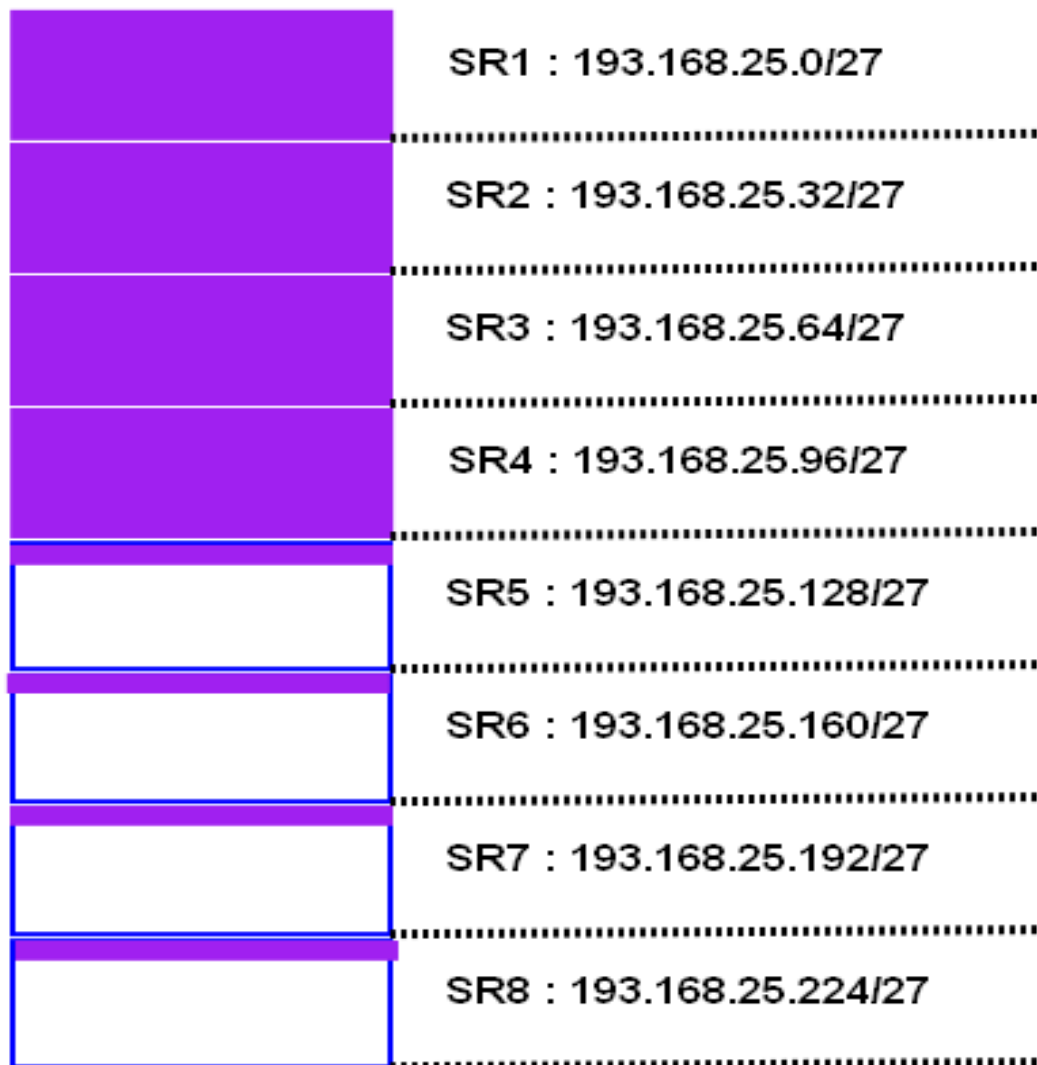


SANS VLSM (4)

Légende

occupé

gaspillé



AVEC VLISM (1)

- On re-divise l'adresse réseau 193.168.25.128/27 en sous-réseaux pouvant contenir 2 machines. → Masque : **255.255.255.252 (/30)**

193.168.25.128/27



193.168.25.100XXX00

{193.168.25.10000000 (128) /30 } : SR5

{193.168.25.10000100 (132) /30 } : SR6

{193.168.25.10001000 (136) /30 } : SR7

{193.168.25.10001100 (140) /30 } : SR8

{193.168.25.10010000 (144) /30 }

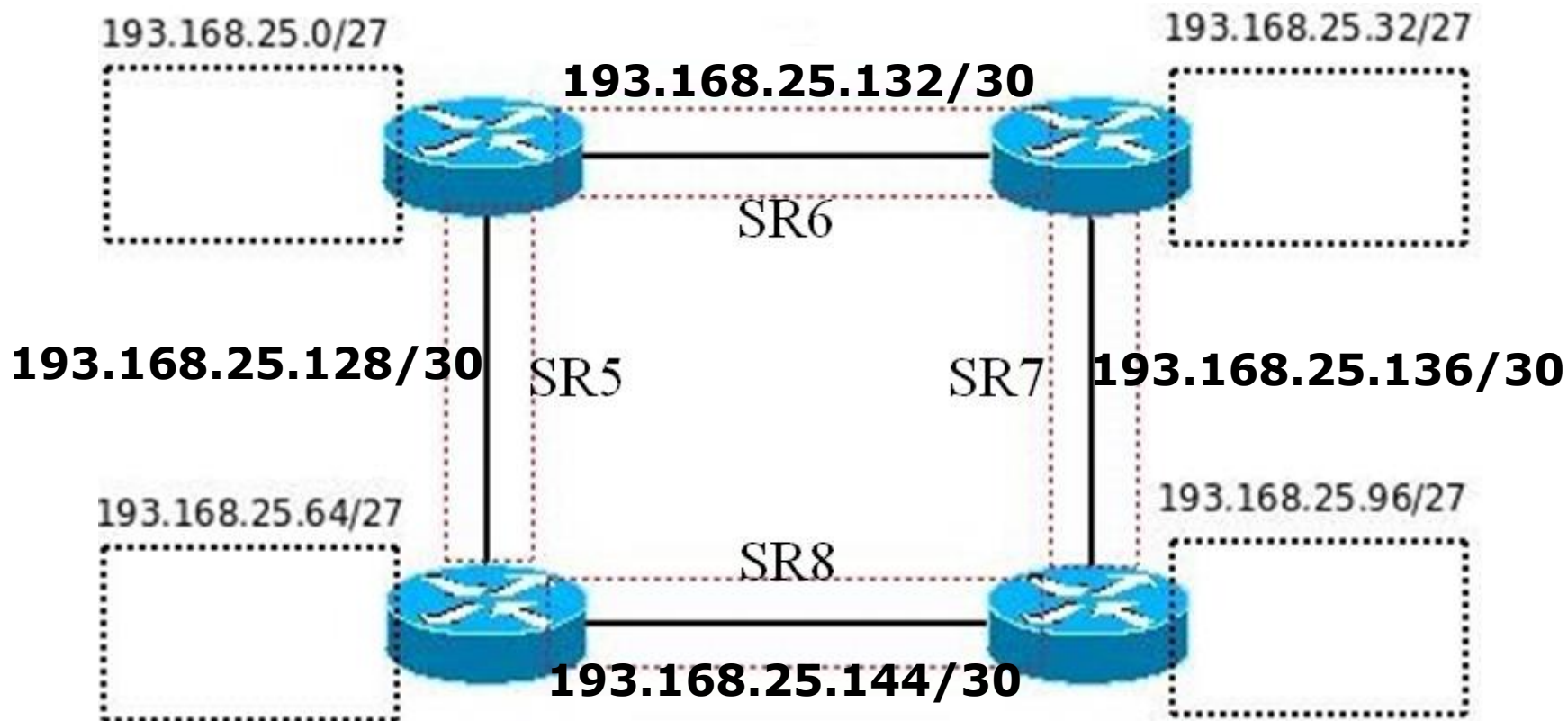
{193.168.25.10010100 (148) /30 }

{193.168.25.10011000 (152) /30 }

{193.168.25.10011100 (156) /30 }

Sur 8 sous-réseaux disponibles ----> 4 vont être utilisés.

AVEC VLSM (2)



AVEC VLSM (3)

Légende

occupé

gaspillé

non-utilisé

SR5 : 193.168.25.128/30

SR7 : 193.168.25.136/30

SR1 : 193.168.25.0/27

SR2 : 193.168.25.32/27

SR3 : 193.168.25.64/27

SR4 : 193.168.25.96/27

SR6 : 193.168.25.132/30

SR8 : 193.168.25.144/30

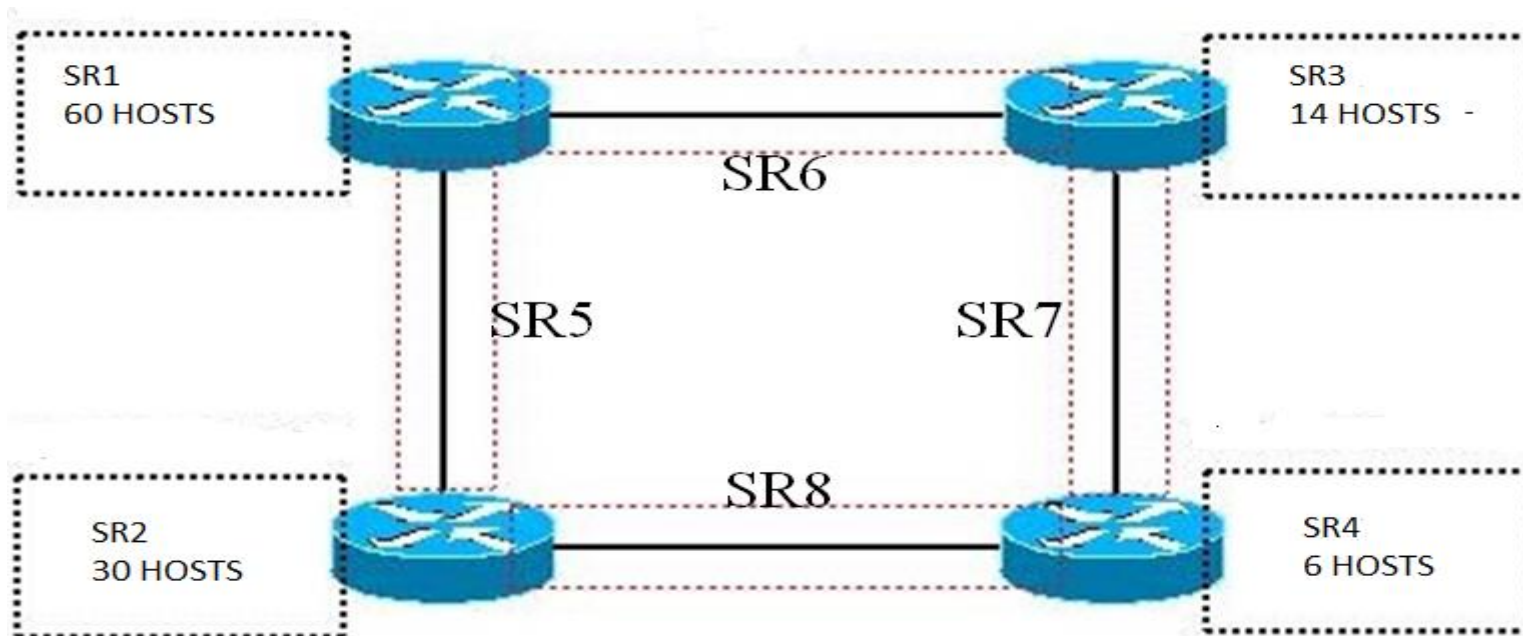
193.168.25.160/27

193.168.25.192/27

193.168.25.224/27

Exemple d'Application

- Exemple: Une entreprise ayant une adresse 10.10.0.0/24 veut utiliser un VSLM pour ne pas gaspiller ses adresses IP.



Donner l'adresse reseau avec son masque sous-reseau sur chaque reseau.

	Reseau	Nbre de Machines	Adresse Reseau	1er machine	Derniere Machine	Adresse de Broadcast
	SR1	60	10.11.48.0/26	10.11.48.1	10.11.48.62	10.11.48.63
	SR2	30	10.11.48.64/27	10.11.48.65	10.11.48.94	10.11.48.95
	SR3	14	10.11.48.96/28	10.11.48.97	10.11.48.110	10.11.48.111
	SR4	6	10.11.48.112/29	10.11.48.113	10.11.48.118	10.11.48.119
	SR5	2	10.11.48.120/30	10.11.48.121	10.11.48.122	10.11.48.123
	SR6	2	10.11.48.124/30	10.11.48.125	10.11.48.126	10.11.48.127
	SR7	2	10.11.48.128/30	10.11.48.129	10.11.48.130	10.11.48.132
	SR8	2	10.11.48.132/30	10.11.48.133	10.11.48.134	10.11.48.135





UNIX

Chapitre 3 : Notion de processus dans le système UNIX

NOTION DE PROCESSUS



Un processus est un programme en cours d'exécution

Un **programme**, produit par un éditeur de liens, est un fichier binaire exécutable mémorisé sur disque.

Pour l'exécuter le système le charge en mémoire, il devient alors un processus .

Un processus est identifié au sein du système par un n° unique, le PID .

Les commandes de gestion de processus:

- Etat des processus actifs : ps
- Arrêter un processus actif : kill
- Différentes façons de lancer un processus (nohup , at time, batch , nice).

AFFICHER L'ETAT DES PROCESSUS ACTIFS

La commande `ps` affiche la liste des processus actifs :

```
$ ps
  PID TTY          TIME COMMAND
 11223 ttyp4        0:00 tcsh
 12609 ttyp4        0:00 ps
 11224 ttyp4        0:02 xv
$
```

ne sont affichés que les processus lancés dans la fenêtre terminal courante.

L'option `-u` permet d'afficher tous les processus appartenant à l'utilisateur spécifié:

```
$ ps -u $USER
  PID TTY          TIME COMMAND
 11222 ?            0:00 hpterm
 11597 ?            0:00 hpterm
 11614 ?            0:00 vuepad
 11126 ?            0:00 hpterm
 11219 tttyp1        0:00 vuepad
10995 ?            0:00 vuesession
 11223 tttyp4        0:00 tcsh
 12610 tttyp4        0:00 ps
 11123 tttyp1        0:00 softmsgsrv
 11598 tttyp5        0:00 tcsh
 11127 tttyp2        0:00 tcsh
 11119 ?            0:13 vuewm
 11128 tttyp2        1:08 netscape
 11220 tttyp2        0:43 xemacs-19.13
 11224 tttyp4        0:02 xv
$
```

sont affichés tous les processus appartenant à l'utilisateur pour la session courante.

La commande ps

Définition

Cette commande affiche la liste des processus actifs sur le système.
Il existe 2 types de processus : les processus systèmes qui accomplissent des services généraux et les processus utilisateurs.
Par défaut, la commande `ps` n'affiche que les processus utilisateurs.

Syntaxe

```
ps [-u utilisateur] [-e] [-f]
```

■ Les options `-ef` affichent les informations complètes sur tous les processus:

<u>UID</u>	<u>PID</u>	<u>PPID</u>	<u>c</u>	<u>STIME</u>	<u>TTY</u>	<u>TIME</u>	<u>COMMAND</u>
root	0	0 0		Jan 1	?	0:06	swapper
root	1	0 0		Jan 9	?	0:00	init
root	2	0 0		Jan 9	?	0:01	vhand
root	3	0 0		Jan 9	?	0:00	statdaemon
root	7	0 0		Jan 9	?	0:00	unhashdaemon



```
root      10      0 0   Jan 9  ?      0:00 syncdaemon
root     417      1 0   Jan 9 console 0:00 /etc/getty -h console console
root     418      1 0   Jan 9  ?      0:00 /usr/vue/bin/vuelogin
root     169      1 0   Jan 9  ?      0:00 /etc/inetd
root     239      1 0   Jan 9  ?      0:00 /usr/local/bin/httpd -d
          /usr/local/httpd
root     237      1 0   Jan 9  ?      1:26 /usr/bin/X11/fs -daemon
ubane11 28693 28417 0 14:01:11 tttyp3 6:49 xemacs
tukalo   3755 28255 0 15:44:42 ttyv2  1:20 /usr/vue/bin/vuepad
tukalo  28255 28254 0 13:47:41 ttyv2  0:00 -tcsh
daemon   419    418 1   Jan 9  ?      79:44 /usr/bin/X11/X :0
root    22060    169 0 09:38:36 ttyv0  0:00 telnetd
ubane11  8180 28593 7 17:20:28 tttyp4  0:00 ps -ef
ubane11 28416 28409 0 13:51:41 ?      0:07 /usr/vue/bin/hpterm
ubane11  7964  7961 0 17:00:38 tttyp5  0:00 tcsh
ubane11 28594 28593 0 13:58:31 tttyp4  0:03 xv
ubane11 28417 28416 0 13:51:41 tttyp3  0:00 tcsh
ubane11   913 28409 0 14:51:45 ?      0:16 /usr/audio/bin/audio_editor
baque    7138      1 0   Jan 15 ?      34:34 xemacs
ubane11  8097 28409 0 17:11:01 ?      0:00 /usr/vue/bin/vuepad
baque   25516 22061 0 10:28:14 ttyv0  0:00 tcsh
root    28277    418 0 13:50:59 ?      0:00 /usr/vue/bin/vuelogin
ubane11 28593 28592 0 13:58:28 tttyp4  0:00 tcsh
root    28254    169 0 13:47:41 ttyv2  0:00 telnetd
ubane11 28592 28409 0 13:58:28 ?      0:02 /usr/vue/bin/hpterm
ubane11 28409 28285 0 13:51:27 ?      0:27 vuewm
ubane11 28285 28277 0 13:51:17 ?      0:00 /usr/vue/bin/vuesession
baque   22061 22060 0 09:38:36 ttyv0  0:00 -tcsh
ubane11  7961 28409 0 17:00:36 ?      0:00 /usr/vue/bin/hpterm
```

ARRETER UN PROCESSUS ACTIF

🔵 La commande `kill` arrête un processus dont on fournit le numéro (PID) :

```
$ ps
  PID TTY          TIME COMMAND
  777 ttyp2        0:00 tcsh
  943 ttyp2        0:00 xv
  964 ttyp2        0:00 ps
$ ps
  PID TTY          TIME COMMAND
  777 ttyp2        0:00 tcsh
  943 ttyp2        0:00 xv
  980 ttyp2        0:00 ps
$ kill 943
[1]      Terminated                  xv
$ ps
  PID TTY          TIME COMMAND
  777 ttyp2        0:00 tcsh
 1026 ttyp2        0:00 ps
$
```

● L'option `-9` permet de forcer la destruction d'un processus :

```
$ ps
  PID TTY          TIME COMMAND
 2571 ttyp2        0:00 vuepad
  777 ttyp2        0:00 tcsh
 2686 ttyp2        0:00 ps
$ kill -9 2571
[1]      Killed                        /usr/vue/bin/vuepad
$ ps
  PID TTY          TIME COMMAND
 2767 ttyp2        0:00 ps
  777 ttyp2        0:00 tcsh
$
```

La commande kill

Définition

Cette commande interrompt un processus en cours d'exécution.
En réalité `kill` envoie un signal au processus spécifié.

Syntaxe

```
kill [-signal] PID
```



Exercice :

- Lancez un éditeur de texte.
- Identifiez son numéro de processus avec la commande `ps`.
- Tuez cet éditeur à l'aide de la commande `kill`.

DIFFERENTES FAÇONS DE LANCER UN PROCESSUS

Laisser vivre un processus même après la fin d'une session (logout) [nohup](#)

Lancer un processus en différé [at time, batch](#)

Lancer un processus avec une priorité particulière [nice](#)

Continuité d'exécution d'un processus : nohup



Définition

Lors de la déconnexion, les processus lancés par l'utilisateur qui s'exécutent encore sont tués automatiquement. Toutefois il est possible de prolonger l'exécution d'un processus en utilisant cette commande.



Syntaxe

```
nohup commande [parametres]
```

Exécution d'un processus en différé : at time, batch



Définition

at : permet de spécifier le moment de l'exécution.

batch : la commande est mise en attente et est exécutée quand le système n'est pas surchargé



Syntaxe

```
at time commande [parametres]
```

```
batch commande [parametres]
```


■ Exemple :

```
$ at 8:15 am Jan 23 mon-prog
```

Exécution d'un processus avec priorité basse : nice



Définition

La commande `nice` permet de donner des priorités plus ou moins élevées pour l'exécution de processus selon l'importance de la tâche qu'ils remplissent.



Syntaxe

```
nice [+/-nombre] commande [parametres]
```

avec 1 (fort) \leq nombre \leq 19 (faible).

Par défaut, nombre est égal à 10.

■ Remarque :

Seul le SU (Super Utilisateur) peut exécuter un processus en augmentant sa priorité. Un utilisateur normal ne peut que la diminuer.

UNIX

Chapitre 4 : Notion d'utilisateurs dans le système UNIX

NOTION D'UTILISATEURS




Tout utilisateur est enregistré dans deux fichiers systèmes :

/etc/passwd

```
$ cat /etc/passwd
root:1id4LZtxn4bTk:0:3:::/bin/sh
daemon:*:1:5:::/bin/sh
bin:*:2:2::/bin:/bin/sh
lp:*:9:7::usr/spool/lp:/bin/sh
nobody:*:-2:60001:uid nobody:/:
vidal:P1rMrwy9FDhzI:201:200:Philippe VIDAL,,,:/users/fudmip/vidal:/usr/local/bin/tcsh
tukalo:Rqg5j8qWg5wk2:203:200:Bruno TUKALO,,,:/users/fudmip/tukalo:/usr/local/bin/tcsh
ubanel1:/YEGKZ31AgWdU:213:200:Isabelle UBANELL,,,:/users/fudmip/ubanel1:/usr/local/bin/tcsh
perrin:q7s1To9hOiNCI:209:203:Marc PERRIN,,,:/users/hp/perrin:/bin/csh
$
```

Ce fichier contient :

- nom de login
- mot de passe chiffré
- numéro unique d'utilisateur (UID)
- numéro unique de groupe (GID)
- nom complet de l'utilisateur
- répertoire initial
- interpréteur de commande 

Le séparateur de champs est le caractère :

/etc/group

```
$ cat /etc/group
root::0:root
bin::2:root,bin
daemon::5:root,daemon
lp::7:root,lp
users::20:root
nogroup::-2:
fudmip:*:200:vidal,tukalo,ubanel1
hp:*:203:perrin
$
```

Ce fichier contient :

- nom de groupe
- numéro unique de groupe (GID)
- liste des utilisateurs du groupe

Remarque : un utilisateur peut faire partie de plusieurs groupes.

Les notions d'utilisateurs et de groupes sont fondamentales pour l'attribution de droits d'accès aux fichiers.

PROTECTION : DROITS D'ACCES

Permissions et contrôle d'accès

Un ensemble de permissions d'accès est associé à tout fichier; ces permissions déterminent qui peut accéder au fichier et comment:

fichier		<u>répertoire</u>	
<u>r</u>	accès en lecture	r	<u>accès en lecture</u>
<u>w</u>	accès en écriture	w	<u>accès en création, modification, destruction</u>
<u>x</u>	accès en exécution	x	<u>accès au nom</u>

Pour accéder à une feuille ou à un noeud dans une arborescence de fichiers, il faut avoir la permission en x sur tous les répertoires de niveau supérieur (chemin d'accès).

Exemple :

Si un utilisateur demande à visualiser le fichier `/users/formation/billy/.login`, il lui faudra avoir le droit en X sur tous les répertoires *traversés*, c'est à dire `users`, `formation`, `billy` ainsi que le droit en lecture sur le fichier `.login`

Il existe trois classes d'utilisateurs : 

<u>u</u>	propriétaire
<u>g</u>	groupe du propriétaire
<u>o</u>	les autres

Il existe quatre principaux types de fichiers :

-	fichier ordinaire
d	répertoire (directory)
c	fichier mode caractère
b	fichier mode bloc

Voyons en détail le résultat de la commande `ls -l`

```

-rw-r--r-- 1 martin fudmip      110 nov 28 14:56 fichier1
drwxr-xr-x 2 martin fudmip    1024 nov 17 13:55 boite
crw-rw-rw- 1 root  sys    57 0x208000 nov 29 16:31 audio
brw-rw-rw- 1 root  sys    7 0x201300 dec 13 1993  tape
  
```

Annotations: 1 points to the first character of permissions, 2 points to the owner permissions, 3 points to the group permissions, 4 points to the other permissions, 5 points to the owner name, 6 points to the group name.

- 1 : type de fichier
- 2 : droits d'accès pour le propriétaire du fichier (5)
- 3 : droits d'accès pour les membres du groupe propriétaire (6)
- 4 : droits d'accès pour tous les autres
- 5 : nom du propriétaire du fichier
- 6 : nom du groupe auquel appartient le fichier

MODIFICATION DES DROITS D'ACCES

Les permissions d'accès peuvent être modifiées par la commande

`chmod mode fichier [ou répertoire]`

Seul le propriétaire du fichier (ou répertoire) a la possibilité de modifier les droits d'accès sur son fichier.

- Contenu du répertoire courant :



```
$ 11
total 2
-rw-rw-r-- 1 ubanell fudmip 19 mars 11 15:21 mon-fich
$
```

- le propriétaire, **ubanell**, a les droits de lecture et écriture, **rw-**, sur le fichier **mon-fich**
- le groupe, **fudmip**, a les droits de lecture et écriture, **rw-**, sur le fichier **mon-fich**
- les autres, ont le droit de lecture, **r--**, sur le fichier **mon-fich**

• J'autorise le groupe **fudmip** à exécuter mon fichier **mon-fich**

```
$ chmod g+x mon-fich
$ ls -l
total 2
-rw-rwxr-- 1 ubanell fudmip 19 mars 11 15:21 mon-fich
$
```

les droits d'accès pour le fichier **mon-fich** ont été changés pour le groupe **fudmip**

• Je supprime le droit de lecture du fichier **mon-fich** pour les autres :

```
$ chmod o-r mon-fich
$ 11
total 2
-rw-rwx--- 1 ubanell fudmip 19 mars 11 15:21 mon-fich
$
```

les autres n'ont plus le droit de lire le fichier **mon-fich**

• Autres exemples

```
chmod o-w fic
```

enlève le droit w aux autres

```
chmod a+x fic
```

rend le fichier exécutable pour tout le monde (a = all)

```
chmod a=rx,u+w fic
```

donne les droits r et x pour tout le monde, ajoute le droit w pour le propriétaire

`chmod u=rw,go=r fic`

donne les droits r et w pour le propriétaire et seulement le droit r pour tout les autres

• Je change les droits d'accès d'un fichier par la valeur octale du mode :

```
$ chmod 644 mon-fich
$ ls -l
total 2
-rw-r--r--  1 ubanell  fudmip           19 mars 11 15:21 mon-fich
$
```


- Pour le propriétaire, autorisation = **6** (en octal), c'est à dire `rw-` soit **110** en binaire.
 - Pour le groupe, autorisation = **4** (en octal), c'est à dire `r--` soit **100** en binaire.
 - Pour les autres, autorisation = **4** (en octal), c'est à dire `r--` soit **100** en binaire.
- Chaque chiffre correspond dans l'ordre aux droits du propriétaire, du groupe et des autres.



★ Exercices sur la commande CHMOD ★



Exercices

- [Exercice 1](#) : modification des droits d'accès en octal
- [Exercice 2](#) : résultat d'une modification des droits d'accès en octal
- [Exercice 3](#) : correspondance des droits d'accès en lettres et en octal
- Exercice 4 :
 - Créez un fichier texte
 - Interdisez tous les accès à ce fichier pour le groupe (en lettres)
 - Autorisez l'accès à ce fichier en lecture au groupe (en octal)
 - Interdisez l'accès à votre Home Directory pour le groupe et les autres 

CHANGEMENT DE PROPRIETAIRE

Le propriétaire d'un fichier ou d'un répertoire peut être changé par la commande :

```
chown NouveauPropriétaire fichier [ou répertoire]
```

Seul le propriétaire du fichier (ou répertoire) et le SU peuvent effectuer ce changement.

- Par exemple, visualisons le contenu de mon répertoire courant

```
$ ls -l
total 2
-rw-r--r--  1 ubanell  fudmip          19 mars 11 15:21 Fich
$
```


p
ropriétaire du fichier **Fich** est **ubanell**

- Je souhaite que l'utilisateur **tukalo** soit propriétaire du fichier **Fich** :

```
$ chown tukalo Fich
$ ls -l
total 2
-rw-r--r--  1 tukalo  fudmip          19 mars 11 15:21 Fich
$
```

le nouveau propriétaire du fichier **Fich** est **tukalo**



Remarque : c'est le propriétaire du fichier **Fich** qui a réalisé l'action .

CHANGEMENT DE GROUPE

Vous pouvez changer le groupe auquel s'applique les protections de niveau groupe par la commande :

```
chgrp NouveauGroupe fichier [ou répertoire]
```

Seul le propriétaire du fichier (ou répertoire) et le SU peuvent effectuer ce changement.

- Contenu de mon répertoire courant :

```
$ ls -l
total 2
-rw-r--r--  1 ubane11  fudmip          19 mars 11 18:07 Fich
$
```

le groupe du fichier **Fich** est **fudmip**

- Je souhaite changer le groupe **fudmip** auquel s'applique les protections de niveau groupe par le groupe **aero** existant :

```
$ chgrp aero Fich
$ ls -l
total 2
-rw-r--r--  1 ubane11  aero          19 mars 11 18:07 Fich
$
```

NOTION DE MASQUE

Tout fichier créé par le système d'exploitation a des droits d'accès par défaut.
Par exemple sous HP-VUE

- un fichier texte a comme droits d'accès **666**,
- un fichier exécutable ou un répertoire a pour droits d'accès **777**.

Pour éviter toute intrusion sur ces fichiers le fichier `.login` ou `.profile` (selon le Shell utilisé) contient souvent la commande `umask 022` qui change les droits d'accès par défaut.

Ces nouveaux droits sont :

- pour un fichier texte, **644**,
- pour un fichier exécutable ou un répertoire, **755**.

Pour les répertoires et les fichiers exécutables, les bits à 1 du masque invalident les droits correspondant.

Inversement, les bits à 0 du masque autorisent les droits correspondant.
Pour les fichiers texte, même application en enlevant en plus le droit en exécution.

Pour obtenir la valeur courante du masque taper la commande `umask`
La valeur retournée est donnée en octal.

Bien entendu, la commande `chmod` ignore les masques



NOTION DE SUPER UTILISATEUR

Les restrictions d'accès s'appliquent aux utilisateurs.

Un seul utilisateur est exempt de contrôles d'accès:

le **Super Utilisateur** ==> login: **root**

(identificateur réservé = 0)

CEPENDANT ...

Les règles énoncées ci-dessus ne suffisent pas! 

Un exemple: *le changement de mot de passe*

- * tout utilisateur doit pouvoir effectuer cette action
- * elle implique un accès en écriture à `/etc/passwd`

- Les utilisateurs sont enregistrés dans :

```
-rw-r--r-- root sys /etc/passwd
```

- Tout utilisateur a la permission de lire `/etc/passwd` mais ne peut donc modifier directement son mot de passe

- Une fonction permet d'effectuer la modification : `/bin/passwd`

```
-rwxr-xr-x root sys /bin/passwd
```

- Avec les permissions mentionnées ci-dessus, l'exécution de `/bin/passwd` s'effectue dans le contexte d'un utilisateur et à son niveau de privilège -> **ECHEC!**

- Seul le propriétaire (ici root) peut effectuer une opération d'écriture