

Systemes d'Exploitation

Licence Informatique 1^{ère} Année

2025-2026

Les systèmes UNIX

Il existe 2 grandes catégories de systèmes UNIX :

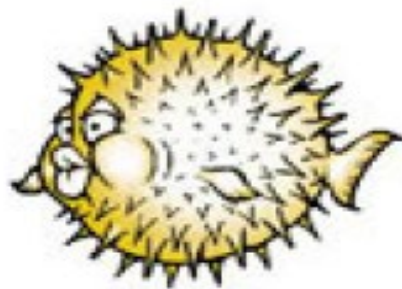
- Les systèmes UNIX propriétaires :
 - Solaris de SUN Microsystems
 - AIX d'IBM
 - MacOS X d'Apple
 - IRIX de Silicon Graphics
 - HP-UX d'Hewlett-Packard
 - ...
- Les UNIX libres.

Les UNIX libres

Ce sont des logiciels libres / Free Software (le code source est disponible, modifiable, redistribuable) divisés en 2 familles :

- des systèmes BSD (***Berkeley Software Distribution***): Les systèmes BSD sont généralement conçus pour être utilisés sur des **serveurs**
 - FreeBSD
 - NetBSD
 - OpenBSD
 - ...
- les systèmes basés sur le noyau Linux :
 - RedHat/Fedora/CentOS
 - Debian
 - Ubuntu
 - OpenSuse
 - ...
- Les systèmes Unix propriétaires ou libres sont tous compatibles entre eux, car respectent la norme POSIX et la Single UNIX specification.

Unix libres



debian



ubuntu

Distribution Linux VS Linux

Une distribution Linux est :

- un système d'exploitation UNIX (basé sur le noyau Linux).
- un ensemble d'applications.

Distribution Linux VS Linux

- Le noyau Linux, fournit en plus :
 - des systèmes de fichiers
 - une pile réseau
 - un très grand nombre de pilotes de périphériques (drivers)

Composants systèmes pour Linux : le projet GNU

- Le projet GNU (GNU is not Unix), a pour but de fournir des utilitaires open-source afin de créer un système d'exploitation complet. Les utilitaires principaux sont :
 - Le compilateur GCC
 - les utilitaires pour la compilation : l'assembleur as, le linker ld, ...
 - le Shell bash
 - la bibliothèque standard C glibc (stdio, stdlib, ...)
 - les utilitaires
 - **texte** (cat, sort, ...),
 - **fichiers** (ls, rm, cp, mv,...),
 - **et shell** (basename, id, ...)
- Le noyau Linux + les utilitaires GNU forment un système d'exploitation UNIX open-source complet.

Première connexion

Créer un compte

- Le système Unix est un système à authentification.
- Pour pouvoir travailler, il faut avoir un **compte** (login).
- Le seul à pouvoir créer un compte est **l'administrateur** (root).
- Pour créer un compte l'administrateur va générer 3 informations :
 - ▶ Le nom sous lequel vous serez connu de la machine : le **login**.
 - ▶ Le mot de passe : **password**.
 - ▶ Un **répertoire d'accueil** : une partie du système de fichiers qui vous sera alloué pour travailler :
 - ★ sur lequel vous aurez tous les droits
 - ★ dont vous serez responsable

C'est le *home directory* ou plus simplement *HOME*.

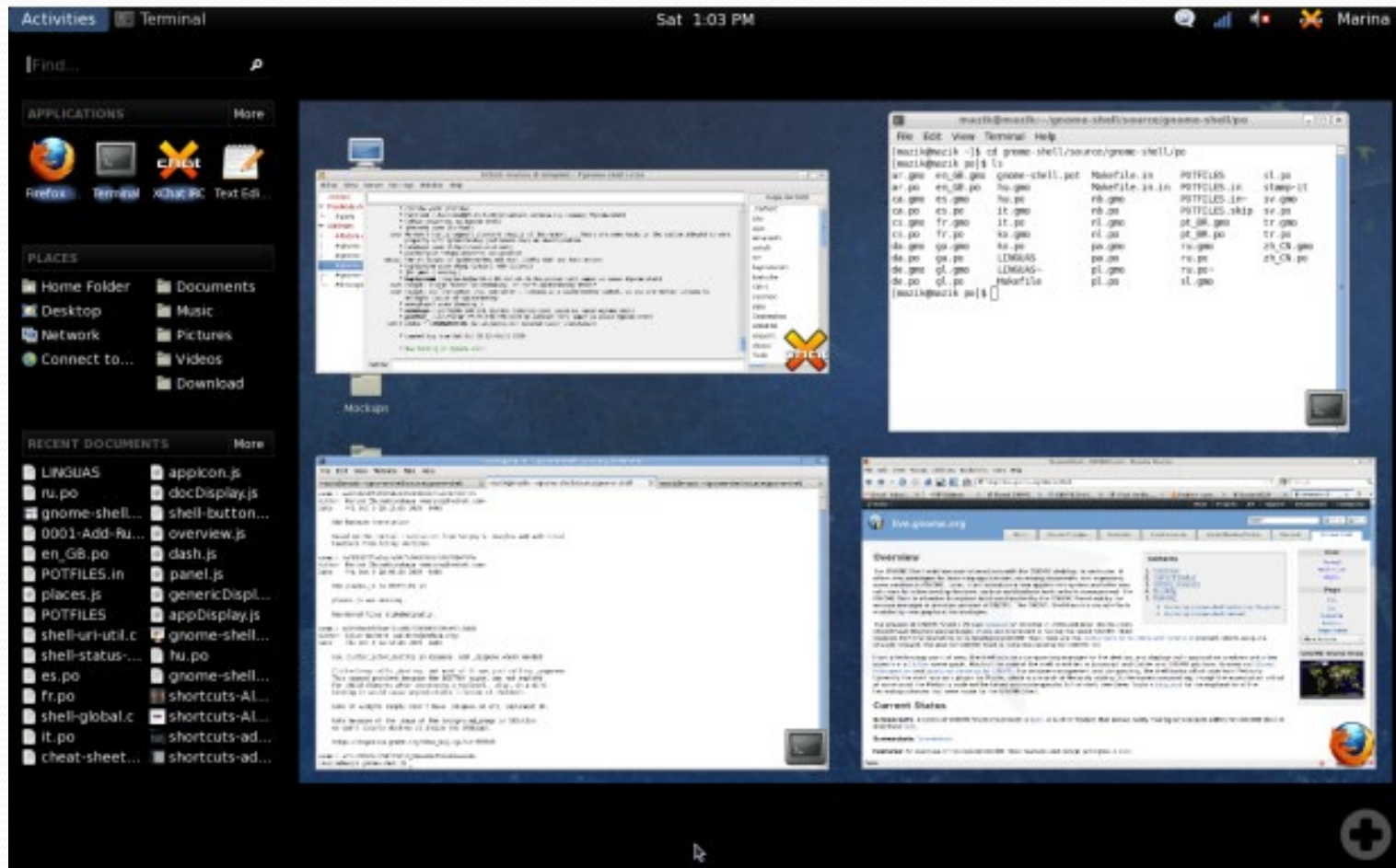
Conventions

- Afin de différencier ce qui est écrit par la machine de ce qui est tapé par l'utilisateur, nous utiliserons la convention suivante :
 - ▶ ce qui est écrit par la machine sera représenté en style normal :
tichit@iml230:~\$
 - ▶ ce qui est tapé par l'utilisateur (les *commandes*) sera représenté en style gras, par exemple : **passwd**.

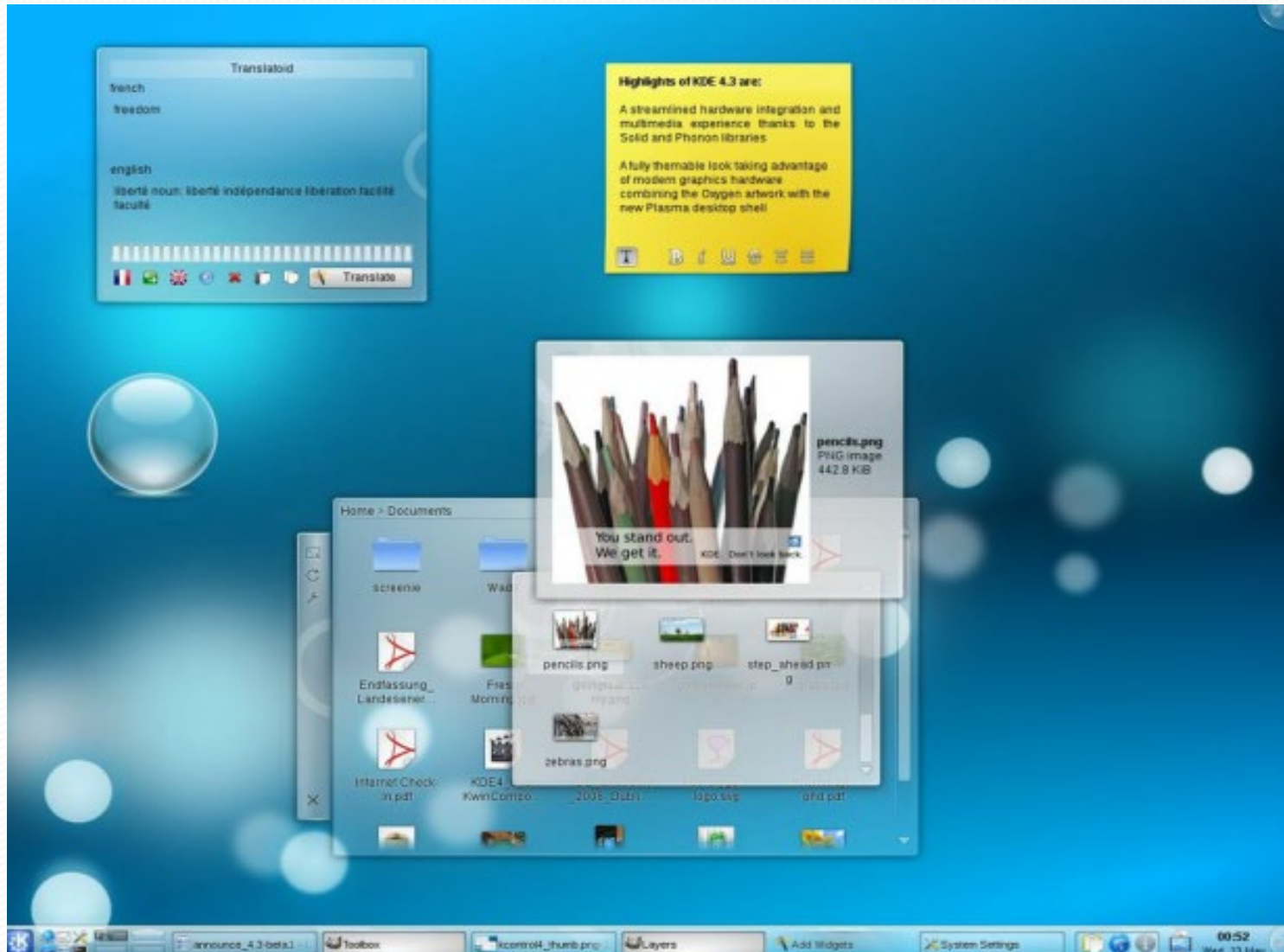
Interface graphique ou texte

- Les systèmes UNIX, comme tout système orienté utilisateur, possèdent une interface graphique avec un gestionnaire de fenêtrage et un environnement de bureau, manipulable à la souris
- Il est cependant plus efficace d'interagir avec le système via le clavier. Vous pouvez ouvrir un Terminal.
- Dans cette fenêtre tourne un interpréteur de commande : le Shell. C'est lui qui permet de lancer des commandes, de les enchaîner, d'automatiser l'exécution d'un certain nombre de tâches.

Gnome 2.32



KDE 4.3

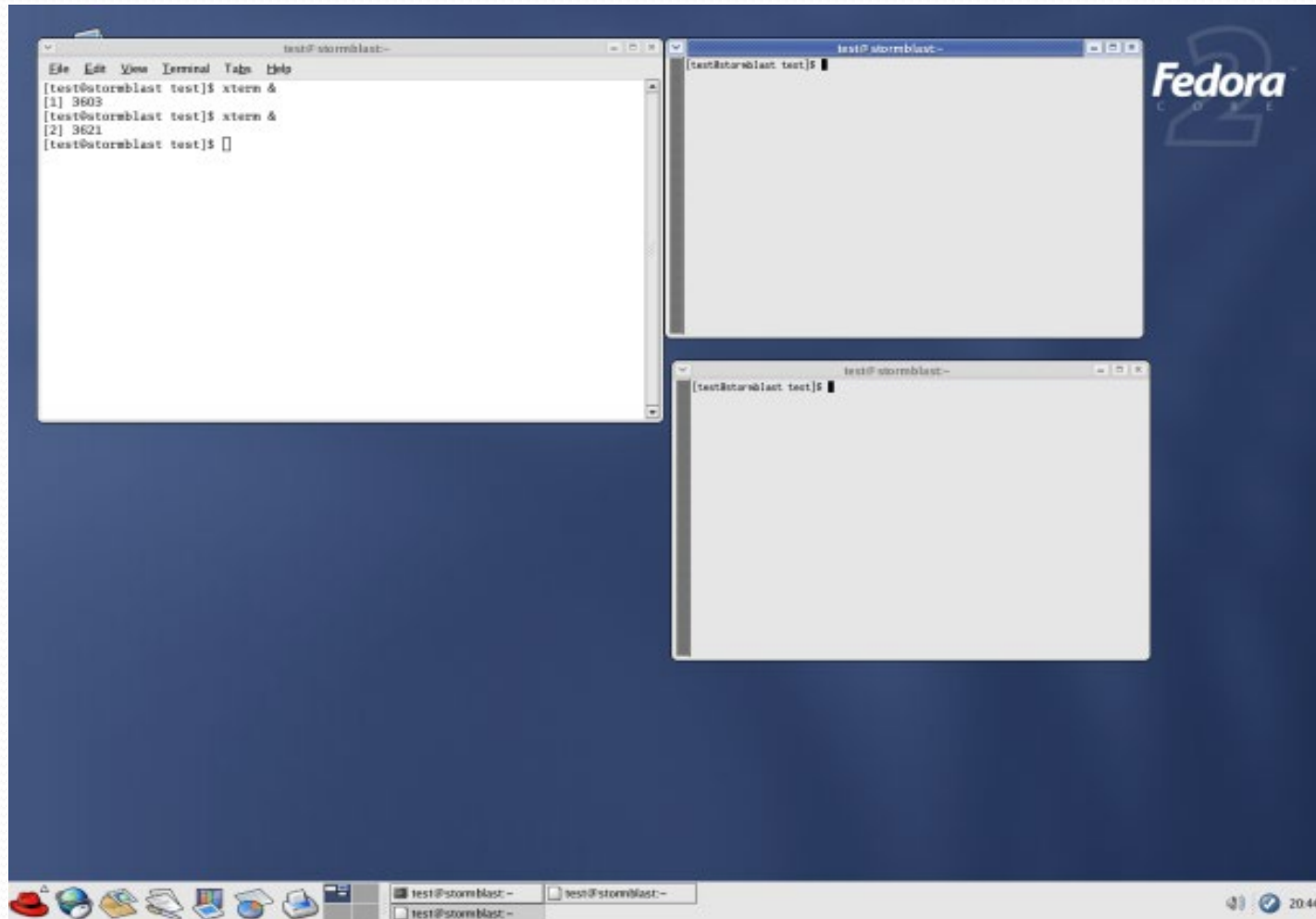


Accès au Shell en mode graphique

Pour exécuter un Shell en mode graphique, on ouvre un *pseudo-terminal* (ou *terminal*, *console*) cherchez dans les menus (ou avec un clic-droit sur le bureau) l'un des programmes suivant :

- Konsole
- Terminal
- Xterm
- LXTerminal

Terminal (xterm)



Connexion en mode texte

Il existe deux types de connexions à un système UNIX :

- les connexions en *mode graphique* (actuellement le défaut)
- les connexions en *mode texte* (ou *mode console*)

Pour passer de l'un à l'autre :

- pour passer en mode texte : **CTRL-ALT-F1**. Après connexion, vous aurez accès directement à l'interpréteur de commandes (le **Shell**).
- pour repasser en mode graphique : **CTRL-ALT-F7**. Après connexion, vous aurez accès à un **environnement de bureau**

Le mode *texte* est davantage utilisé par les *administrateurs* qui connaissent l'*interface en ligne de commande* (le *Shell*). Il est aussi utilisé quand on a un problème de *driver graphique*.

Shell

Le Shell

- Le shell est un langage interprété.
 - Erreurs peuvent être facilement localisables.
 - Modifications faciles.
 - Pas de compilation ou d'édition de liens.
- Le shell manipule, à la base, des chaînes de caractères.
 - Raisonnement uniforme en termes de chaînes.
- Le shell est adapté au prototypage rapide.
 - Communication entre processus.
 - Substitutions de commandes et de variables.
- Le shell est un langage glu .
 - Connexion de composants écrits dans des langages différents.

Environnement de travail

Grâce au Shell il est possible

- De rappeler des commandes au moyen de l'historique
- D'interagir avec la commande courante
- De gérer des travaux lancés en arrière-plan
- D'initialiser les variables d'environnement (configuration)
- D'exécuter un *pipeline* de commandes

On exécute une commande

- en mode interactif ; ou
- par l'intermédiaire d'un *script*

L'interpréteur de commandes : le Shell

L'interpréteur de commandes effectue les opérations suivantes :

- Ecriture sur l'écran d'une invite de commande : le **prompt**.
tichit@iml230:~\$
- Lecture au clavier d'une ligne tapée par l'utilisateur et se terminant par un *return*.
tichit@iml230:~\$ **cal 6 2009**
- Interprétation de la commande, le premier mot étant le nom de la commande désirée, les autres étant ses arguments.
- Exécution de la commande désirée et affichage des résultats.

```
      juin 2009
di lu ma me je ve sa
      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
```

- Retour à la première étape.

Commandes

Commandes

- Il existe des centaines de commandes. Exécutez-en quelques-unes :
 - **echo [argument]** : écriture sur la sortie standard.
tichit@iml230:~\$ **echo Bonjour Laurent**
Bonjour Laurent
 - **echo** sans argument : affiche un retour à la ligne.
tichit@iml230:~\$ **echo**
 - **cat [fichier...]** : liste le contenu des fichiers passés en paramètre.

Syntaxe d'une commande

Rappelons la syntaxe générale d'une commande.

```
[ chemin/]nom_cmd [ option ... ] [ argument ... ]
```

- Suite de mots séparés par un ou plusieurs séparateurs.
- Un séparateur est une tabulation horizontale ou un espace.
- Un mot est une suite de caractères non blancs.
- Plusieurs caractères ont une signification spéciale pour le Shell : ce sont les méta-caractères (ex : |, <).

Une commande comporte généralement un certain nombre de paramètres spécifiques appelés **options**. Elles permettent de modifier le comportement de base de la commande. Les options sont indiquées par le caractère -.

Exemple : `ls -l`

La complétion

Sous le *Shell* La touche <TAB> (tabulation) permet d'effectuer une complétion.



Le premier mot est **toujours** supposé être une *commande*, les suivant, des chemins vers des *fichiers* accessibles. Exemples :

- tichit@iml230:~\$ **ec**<TAB>
echo
- tichit@iml230:~\$ **da**<TAB>
date
- tichit@iml230:~\$ **who**<TAB><TAB>
who whoami
- tichit@iml230:~\$ **whoa**<TAB>
whoami

Le Manuel

- L'aide en ligne : la commande **man** : Elle permet de consulter le manuel du shell à propos d'une commande. Elle offre une documentation sur les commandes Unix, les fonctions C, Perl...
- Exemple:
tichit@iml230:~\$ **man 'nom de commande'**

Le Manuel

- Les sections traditionnelles du manuel sont :

- ❶ **Commandes usager** : commandes utilisateur Unix.
- ❷ **Appels systèmes** : interface entre les programmes et le système d'exploitation.
- ❸ **Fonctions de bibliothèques** : fonctions C qu'un programme peut appeler.
- ❹ **Périphériques** : interfaces avec les différentes ressources matérielles.
- ❺ **Fichier** : formats de fichier les plus importants.
- ❻ **Jeux** : jeux standards de Unix.
- ❼ **Divers** : informations diverses (impression, édition).
- ❽ **Administration** : commandes d'administration.

Le Manuel - options

Pour la commande **man** :

- L'option **-f** permet d'obtenir un rapide survol de l'action d'une commande :

```
tichit@iml230:~$ man -f nom-de-commande
```

- L'option **-k** permet d'obtenir les différentes pages de manuel contenant un mot clé :

```
tichit@iml230:~$ man -k mot-clé
```

Quelque commande shell

- Lister des fichiers dans un répertoire : `ls`

- `ls` : donne la liste des fichiers et répertoires du répertoire courant ;
- `ls rep` : idem pour le répertoire *rep* ;
- `ls a*` : donne la liste des fichiers commençant par 'a'.

Options utiles :

- `ls -l` : liste tous les arguments des fichiers (taille, date, propriété ...) ;
- `ls -t` : liste les fichiers par date de dernière modification ;
- `ls -a` : liste tous les fichiers, y compris ceux dont le nom commence par un '.' ;
- `ls -R` : affiche aussi le contenu de tous les sous-répertoires (récursivement).

Quelque commande shell

- Se déplacer dans les répertoires : **cd**

- `cd` : sans argument, retourne dans le répertoire principal (identique à `cd ~`);
- `cd rep` : pour aller dans le sous-répertoire *rep*;
- `cd ..` : pour remonter dans le répertoire parent ;
- `cd rep/sous_rep` : pour aller dans le sous répertoire *sous_rep* du répertoire *rep*.

Les répertoires peuvent être désignés à l'aide de leurs chemins d'accès (*path* en anglais) absolu (c'est-à-dire depuis la racine) ou relatif (au répertoire courant).

Quelque commande shell

Savoir où on est : **pwd**

- La commande `pwd` (print working directory) retourne le chemin absolu du répertoire courant.

Quelque commande shell

Créer un répertoire : **mkdir**

- La commande mkdir (make directory) permet de créer un répertoire.
- – mkdir toto : crée le répertoire toto dans le répertoire courant ;
- – mkdir ../toto : idem dans le répertoire parent.

Quelque commande shell

- Détruire un fichier : **rm**

La commande `rm` (*remove*) permet de supprimer des fichiers ou des répertoires.

`rm toto.txt` détruit le fichier en question. **ATTENTION** : un fichier supprimé ne peut plus être retrouvé.

Options

-i : demande une confirmation ;

-R ou -r : pour supprimer récursivement toute une hiérarchie de fichiers.

Doit être utilisé avec précaution.

- Détruire un répertoire vide : **rmdir**

La commande `rmdir` (*remove directory*) ne permet de détruire un répertoire que s'il est vide. Pour détruire un répertoire non vide, utiliser `rm -R`.

Quelque commande shell

- Déplacer/copier un fichier : **mv/cp**

La commande `cp` (*copy*) permet de dupliquer un fichier en le renommant éventuellement.

`mv` (*move*) est utilisée dans les mêmes conditions que `cp` pour déplacer un fichier, en le renommant éventuellement.

Quelque commande shell

Visualiser un fichier : **cat/less/more/head/tail**

- Ces quatre commandes permettent de visualiser le contenu d'un fichier ASCII. Attention, ce ne sont pas des éditeurs, vous ne pouvez donc pas modifier les fichiers. Le mode de fonctionnement est un peu différent :
 - `cat` fait défiler tout le contenu du fichier du début à la fin. La commande `cat` sert également à *concaténer* deux fichiers, c'est-à-dire à les joindre (voir plus loin).
 - `less` fait défiler le contenu écran par écran. À utiliser pour des fichiers longs. Pour afficher une ligne de plus : touche Entrée. Pour afficher un écran de plus : barre d'espace. Pour interrompre la commande : touche q.
 - `more` presque identique à `less`, en moins bien.
 - `head` affiche le début du fichier (nombre de lignes en option).
 - `tail` affiche la fin du fichier (nombre de lignes en option).

Quelque commande shell

Comparer deux fichiers : **diff**

- `diff file1 file2` : compare les deux fichiers, et liste leurs différences ligne à ligne. Consulter le manuel (`man diff`) pour les options.

Quelque commande shell

- Pour modifier les droits d'accès à un fichier : **chmod**

`chmod x+y chemin d'accès à un fichier ou à un répertoire` : pour ajouter des droits.

`chmod x-y chemin d'accès à un fichier ou à un répertoire` : pour supprimer des droits.

x prend ses valeurs parmi ugoa (u pour user, g pour group, o pour others, a pour all) et y prend ses valeurs parmi rwx (r pour read, w pour write, x pour exécuter).

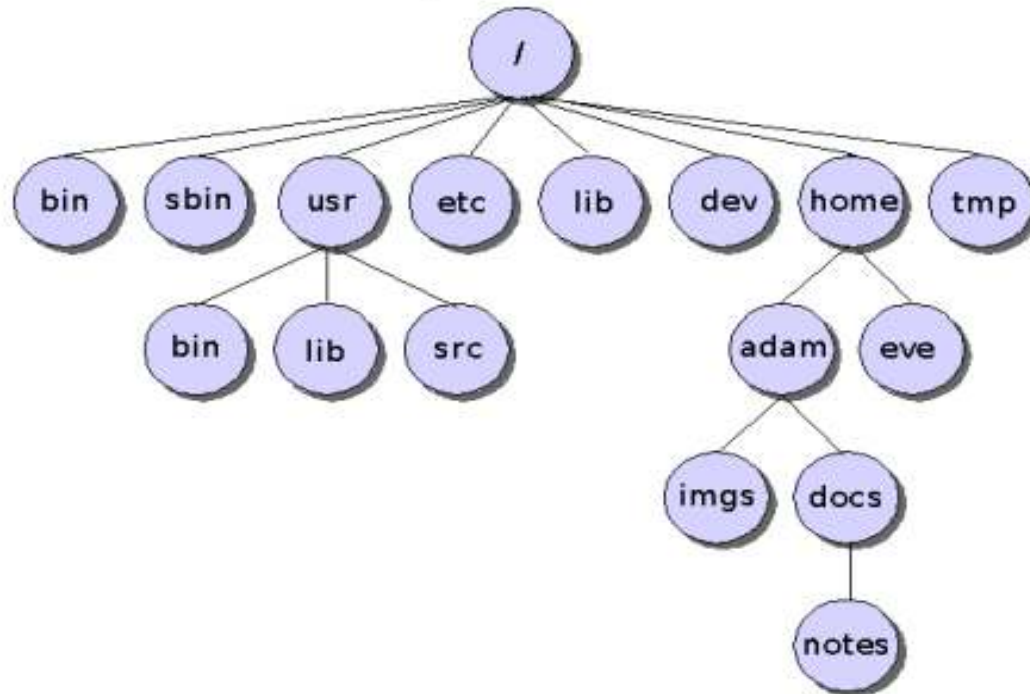
Quelque commande shell

- Retrouver une chaîne de caractère dans un fichier : **grep**
- Cette commande permet d'identifier un fichier contenant une chaîne de caractère donnée :
grep -e string filename donne la liste des occurrences de la chaîne de caractère string dans le fichier filename.

Structure du système de fichiers

Structure du système de fichiers

- La hiérarchie des fichiers est représentée sous forme d'arborescence.



- Le repertoire `/` est celui qui englobe tous les autres. Il s'agit de la **racine**.
- Le chemin absolu du fichier *notes* est `/home/adam/docs/notes`

Répertoires spéciaux

- Le répertoire courant ou répertoire de travail (working directory) est “celui où l’on se trouve”.
- Se déplacer dans le système de fichier revient à changer de répertoire courant.
- On utilise la commande : `cd nom-de-répertoire` (pour change directory).
- Pour connaître le chemin absolu du répertoire de travail : `pwd` (pour path working directory)
 - `tichit@iml230:~$ cd Test`
 - `tichit@iml230:~/Test$ pwd`
 - `/home/tichit/Test`
- Attention : l'accès à un fichier ne peut s'effectuer que vers un fichier visible à partir du répertoire de travail.

Le répertoire d'accueil

- Chaque usager dispose d'un répertoire d'accueil (home directory) qui lui appartient.
- Le nom de ce répertoire est contenu dans la variable HOME.
- La commande `cd` permet de revenir dans son répertoire d'accueil.
 - `tichit@iml230:~$ cd /etc`
 - `tichit@iml230:/etc$ pwd`
 - `/etc`
 - `tichit@iml230:/etc$ cd`
 - `tichit@iml230:~$ pwd`
 - `/home/tichit`
- Sous le Bash, `~` est un alias pour `/home/tichit`.

Les répertoires . et ..

- Tout répertoire contient deux répertoires particuliers qui ont pour noms . et ..
- . représente le répertoire de travail (répertoire courant),
- .. représente le répertoire parent du répertoire courant.
- Pour remonter dans le répertoire parent : `cd ..`

Commandes du système de fichiers

Lister les fichiers

- Lister le nom des fichiers du répertoire courant : `ls`
(pour list) `tichit@iml230:~/LaserBeam$ ls`
- `ls -a` permet de visualiser les fichiers cachés par défaut.
Tout fichier dont le nom commence par “.” est caché (.,
..., fichiers de configuration)
- Tapez `ls`, puis `ls -a` dans votre répertoire d'accueil.

Supprimer des fichiers

- Détruire un fichier : `tichit@iml230:~/LaserBeam$ rm [fichier]`
(pour remove)
- Attention, sous Unix, par défaut, un fichier détruit ne peut pas être récupéré.

Informations associées aux fichiers

Des informations spécifiques sont attachées aux fichiers :

```
tichit@iml230:~/Test$ ls -l
```

```
total 36
```

```
-rw-r--r-- 1 tichit tichit 9145 2009-04-03 11:44 Test.txt  
-rw-r--r-- 1 tichit tichit 21702 2009-04-03 11:50 Titi.pl
```

Les informations sont dans l'ordre :

- Type du fichier et protections
- Nombre de liens sur le fichier
- Le nom du propriétaire
- Le nom du groupe auquel appartient le propriétaire
- Taille exprimée en octets
- Date et heure de la dernière modification
- Nom du fichier

Copie, déplacement, renommage

- Afficher le contenu d'un fichier sur la sortie standard (le terminal actif) : **cat fichier** (pour *catalog*).

```
tichit@iml230:~ cat HelloWorld.pl
```

```
#!/usr/bin/perl  
print("Hello World!\n");
```

- Dupliquer un fichier : **cp** (pour *copy*).

```
tichit@iml230:~/Test$ ls
```

```
Test.txt
```

```
tichit@iml230:~/Test$ cp Test.txt Toto.txt
```

```
Test.txt Toto.txt
```

- Renommer/Déplacer d'un fichier : **mv** (pour *move*).

```
tichit@iml230:~/Test$ mv Toto.txt Titi.pl
```

```
Test.txt Titi.pl
```

Création de fichier, type du fichier

- **touch fichier :**

- ▶ Modifie la date de dernier accès au fichier
- ▶ Crée le fichier s'il n'existe pas.

```
tichit@iml230:~$ ls -l
```

```
-rw-r--r-- 1 tichit tichit 21702 2009-04-03 11:50 Titi.pl
```

```
tichit@iml230:~$ touch Titi.pl
```

```
tichit@iml230:~$ ls -l
```

```
-rw-r--r-- 1 tichit tichit 21702 2009-04-03 12:41 Titi.pl
```

- **file [-f] fichier :** Décrit le contenu du fichier.

```
tichit@iml230:~$ file script
```

```
script: Bourne-Again shell script text executable
```

Création/destruction d'un répertoire

- **mkdir répertoire...** : crée des répertoires.

```
tichit@iml230:~/Test$ mkdir rep1
```

```
tichit@iml230:~/Test$ ls
```

```
rep1/ Test.txt Titi.pl
```

- **rmdir répertoire...** : supprime des répertoires (vides!).

```
tichit@iml230:~/Test$ rmdir rep1
```

```
tichit@iml230:~/Test$ ls
```

```
Test.txt Titi.pl
```

- Pour supprimer un répertoire et son contenu :

rm -fr répertoire (r = recursive ; f=force).

ATTENTION quand vous utilisez **rm** !

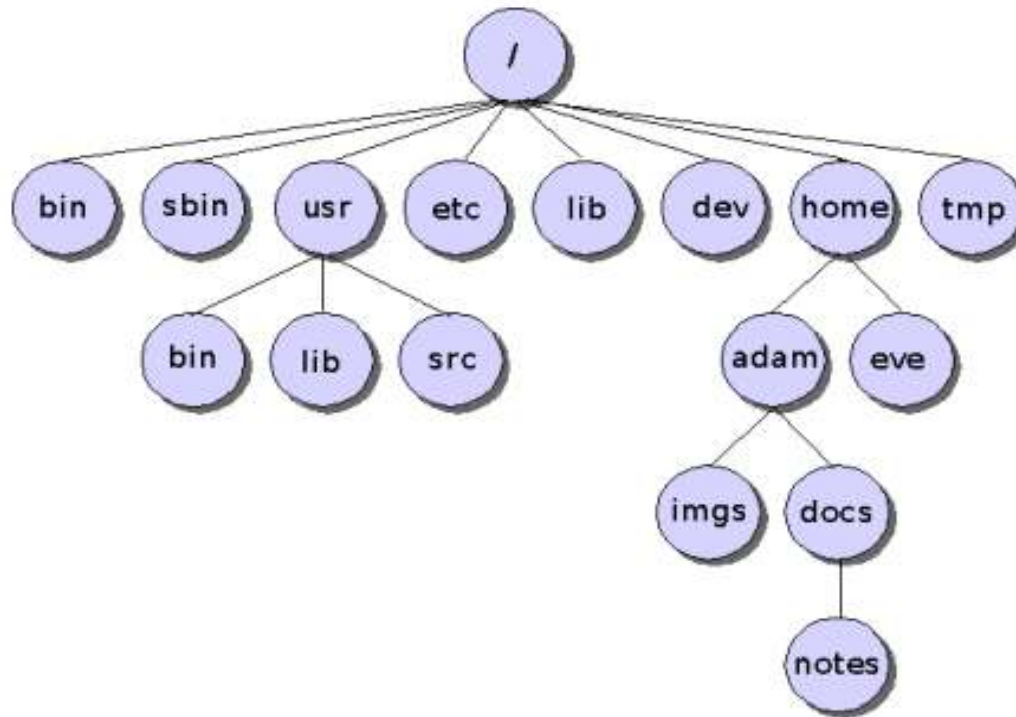
Chemins

Chemin des fichiers

Jusqu'à présent nous avons travaillé sur les fichiers du répertoire de travail. On peut travailler de la même façon sur n'importe quel fichier du système.

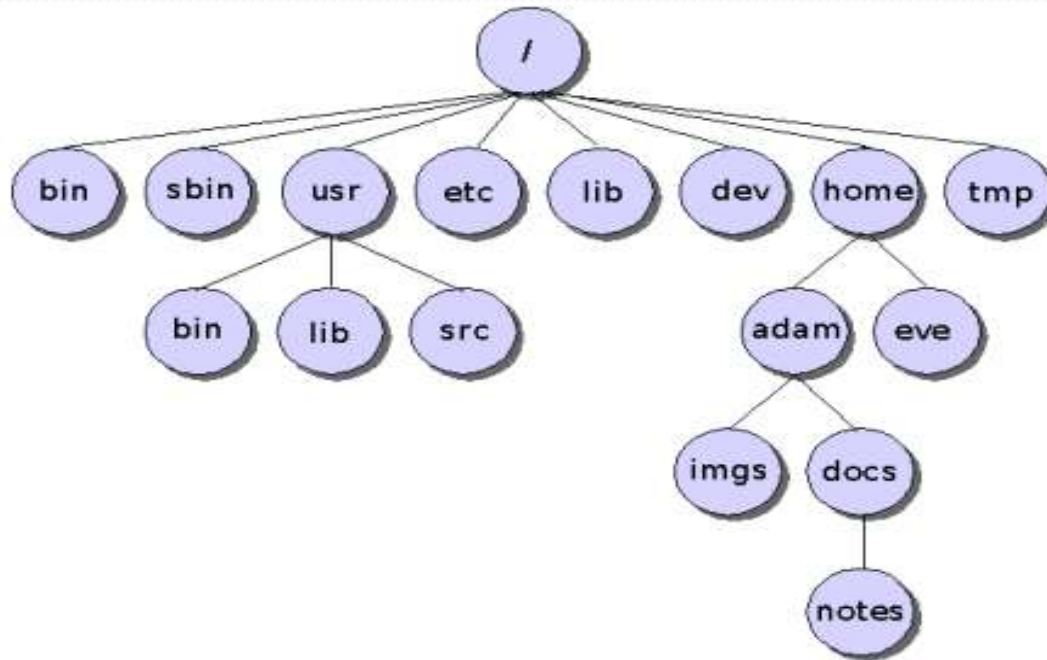
- On peut indiquer son chemin **absolu** : chemin à partir de la racine (/) du système de fichiers.
- On peut indiquer son chemin **relatif** : chemin à partir du **répertoire de travail**.

Chemin absolu



- Le chemin absolu du fichier *notes* est : **/home/adam/docs/notes**
- Le chemin absolu du répertoire *eve* est : **/home/eve**

Chemin relatif



- Le chemin relatif du fichier *notes* à partir du répertoire *adam* est : **docs/notes**
- Le chemin relatif du fichier *notes* à partir du répertoire *eve* est : **../adam/docs/notes**

Chemin des fichiers

- Plutôt que de recourir tout le temps à la commande **cd**, il est plus rapide de se servir des chemins.

- Ex : Préférer :

```
tichit@iml230:~/Test$ cat /etc/fstab
```

à

```
tichit@iml230:~/Test$ cd /etc
```

```
tichit@iml230:~/Test$ cat fstab
```

On a aussi :

- ```
tichit@iml230:~/Test$ mv ~/Desktop/image.jpg ~/Images/
```
- ```
tichit@iml230:~/Test$ mv ~/Desktop/image.jpg  
~/Images/image1.jpg
```
- ...

Chemin des fichiers

Pour obtenir le nom d'un fichier à partir de son chemin complet, on utilise la commande suivante :

- **basename chaîne suffixe** : supprime le plus long préfixe se terminant par '/' de la chaîne et le suffixe (s'il se trouve en fin de chaîne).
tichit@iml230:~\$ **basename /home/tichit/Test/test.txt**
test.txt
- **dirname chemin-de-fichier** : donne le chemin du répertoire dans lequel se trouve le fichier. tichit@iml230:~\$ **dirname /home/tichit/Test**
/home/tichit

Commandes relatives au système de fichiers

- **du [option] fichier-ou-répertoire** : place disque utilisée par un fichier/répertoire en kilo-octets.

```
tichit@iml230:~$ du Test
```

```
40 Test/
```

- **df [répertoire]** : place libre et occupée sur le(s) système(s) de fichiers.

```
tichit@iml230:~$ df .
```

Sys. de fich.	1K-blocs	Occupé	Disponible	C
/dev/sda6	14136684	9092060	4326524	68% /t

Note : . représente le répertoire courant.

Utilisateurs, Groupes, Permissions

Les utilisateurs et les groupes

Un usager possède :

- Un login
- Un numéro d'identification (UID, *User IDentification*)
- Un nom et prénom
- Un mot de passe
- Un nom de groupe principal
- Un numéro de groupe d'appartenance (GID, *Group IDentification*)
- Un répertoire d'accueil
- Un shell par défaut

De plus :

- Unix permet de regrouper des utilisateurs en groupes.
- Un groupe possède certains privilèges.
- Un utilisateur peut appartenir à plusieurs groupes.

Root

Il existe un compte spécial :

- login = root
- group = root
- HOME = /root
- UID = 0
- GID = 0

Ce compte à tous les droits. C'est le *super-utilisateur*. Les administrateurs de la machine ont accès à ce compte.

Les utilisateurs et les groupes

- Les informations relatives aux utilisateurs sont stockées dans le fichier **/etc/passwd**.

```
tichit@iml230:~$ cat /etc/passwd | grep tichit
```

```
tichit:x:1000:1000:Tichit Laurent,,,:/home/tichit:/bin/bash
```

- La commande **id** permet de connaître les informations relatives à un utilisateur :

```
tichit@iml230:~$ id
```

```
uid=1000(tichit) gid=1000(tichit) groupes=1000(tichit),  
4(adm),20(dialout),21(fax),24(cdrom),25(floppy),  
26(tape),29(audio),30(dip),44(video),46(plugdev),  
104(scanner),112(netdev),113(lpadmin),115(powerdev),  
117(admin),119(fuse)
```

Les utilisateurs et les groupes

- Les informations relatives aux groupes sont stockées dans le fichier **/etc/group** :

```
tichit@iml230:~$ cat /etc/group | grep tichit
```

Root

- La commande **su [login]** permet de devenir *root* (ou l'utilisateur *login*).
- Sur les distribution Linux récentes, le compte *root* est parfois désactivé.

Dans ce cas, on se servira de la commande **sudo [login] commande**, au cas pas cas, qui permet de devenir *temporairement root* (ou l'utilisateur *login*) pour exécuter la commande.

Droits d'accès

Les protections et les privilèges

3 manières d'accéder à un fichier

- Lire son contenu (Read)
- Modifier son contenu (Write)
- S'il s'agit d'un programme, l'exécuter (eXecute)

3 catégories d'accès à un fichier

- Le propriétaire du fichier (User)
- Les membres du groupe auquel appartient le fichier (Group)
- Tous les autres usagers (Other)

⇒ 9 protections pour un fichier donné :

- lecture pour son propriétaire
- écriture pour son propriétaire
- exécution pour son propriétaire
- lecture pour les membres du groupe du fichier
- écriture pour les membres du groupe du fichier
- exécution pour les membres du groupe du fichier
- lecture pour les autres usagers
- écriture pour les autres usagers
- exécution pour les autres usagers

Visualisation des protections

La commande **ls -l** permet de connaître les droits d'un fichier :

```
-rwxrwxrwx  1 tichit enseig 826 2009-04-21 12:13 Unix.pl  
-rwxr-x--x  1 tichit enseig 826 2009-04-21 12:13 Unix.sh
```

Les 9 lettres `rwxrwxrwx` et `rwxr-x--x` symbolisent dans l'ordre :

- Les 3 droits associés au propriétaire
- Les 3 droits associés aux membres du groupe auquel appartient le fichier
- Les 3 droits associés aux autres usagers
- **r** indique une autorisation de lire
- **w** indique une autorisation d'écrire
- **x** indique une autorisation d'exécuter (ou accéder)
- **-** indique une interdiction
- *Unix.pl* est donc lisible, modifiable et exécutable par tous.
- *Unix.sh* est lisible par l'utilisateur *tichit* et les membres du groupe *enseig*, modifiable par l'utilisateur *tichit* uniquement, exécutable par tout le monde.

Visualisation du type de fichier

Le premier caractère fourni par `ls -l` indique le type de fichier

- - Fichier standard
- **d** Répertoire
- **p** Pipe (tube nommé)
- **l** Lien symbolique
- **s** Socket
- **c** Périphérique en mode caractère (clavier)
- **b** Périphérique en mode bloc (disque dur)

```
-rw-r--r-- 1 tichit tichit 2912 2009-03-03 11:10 Pub.doc  
drwxr-xr-x 2 tichit tichit 4096 2009-04-22 14:55 unix
```

Pub.doc est un fichier standard, *unix* un répertoire.

Mise en place des permissions

Seul le propriétaire du fichier peut modifier les droits d'accès à un fichier par la commande

chmod mode nom_de_fichier.

On peut modifier les permissions de 2 manières :

- de manière absolue (sans considérer les permissions actuelles)
- relatives (en se modifiant les permissions actuelles)

Permissions absolues

Elles sont exprimées en octal. Si l'on désire uniquement que :

- Le fichier soit lisible
 - ▶ Par le propriétaire → 400
 - ▶ Par les membres du groupe → 040
 - ▶ Par les autres → 004
- Le fichier soit modifiable
 - ▶ Par le propriétaire → 200
 - ▶ Par les membres du groupe → 020
 - ▶ Par les autres → 002
- Le fichier soit exécutable
 - ▶ Par le propriétaire → 100
 - ▶ Par les membres du groupe → 010
 - ▶ Par les autres → 001

Permissions absolues

Bien entendu, on peut composer les permissions. Il suffit d'additionner les valeurs.

- **chmod 644 fichier.txt** : propriétaire peut lire et modifier, groupe et autre peuvent uniquement lire.
- **chmod 755 fichier.pl** : idem plus tous peuvent exécuter.
- **chmod 710 fichier.sh** : le propriétaire a tous les droits, les membres du groupe peuvent uniquement exécuter, les autres n'ont aucun accès.

Permissions relatives (Symbolique)

En symbolique le mode est de la forme :

chmod “qui” “quelle action” “quelle permission” fichier
“Qui”

- **u** : user
- **g** : group
- **o** : other
- **a** : all

“Quelle action”

- **+** : ajoute des permissions
- **-** : retire des permissions

Ex :

- **chmod o-rw fichier.txt :**

Permissions relatives (Symbolique)

En symbolique le mode est de la forme :

chmod “*qui*” “*quelle action*” “*quelle permission*” **fichier**
“*Qui*”

- **u** : user
- **g** : group
- **o** : other
- **a** : all

“*Quelle action*”

- **+** : ajoute des permissions
- **-** : retire des permissions

Ex :

- **chmod o-rw fichier.txt** : enlève les permissions en lecture et écriture pour les autres.
- **chmod a+x fichier.pl** :

Permissions relatives (Symbolique)

En symbolique le mode est de la forme :

chmod “qui” “quelle action” “quelle permission” fichier

“Qui”

- **u** : user
- **g** : group
- **o** : other
- **a** : all

“Quelle action”

- **+** : ajoute des permissions
- **-** : retire des permissions

Ex :

- **chmod o-rw fichier.txt** : enlève les permissions en lecture et écriture pour les autres.
- **chmod a+x fichier.pl** : donne à tous le droit d'exécuter.
- **chmod go-w fichier.sh** :

Permissions relatives (Symbolique)

En symbolique le mode est de la forme :

chmod “qui” “quelle action” “quelle permission” fichier
“Qui”

- **u** : user
- **g** : group
- **o** : other
- **a** : all

“Quelle action”

- **+** : ajoute des permissions
- **-** : retire des permissions

Ex :

- **chmod o-rw fichier.txt** : enlève les permissions en lecture et écriture pour les autres.
- **chmod a+x fichier.pl** : donne à tous le droit d'exécuter.
- **chmod go-w fichier.sh** : enlève aux membres du groupe et aux autres le droit de modifier.

Permissions

Pour changer le propriétaire d'un fichier (ou le groupe propriétaire) :

- **chown nom fichier**
- **chgrp groupe fichier**

Permet de changer le nom (uid) ou le groupe (gid) d'appartenance du fichier.

Exécuté principalement par le super-utilisateur.

Les Inodes et les liens

Les inodes

- A tout fichier est associé une structure de données (sur disque) appelée Inode (*information node*).
- Dans un inode se trouvent différentes informations :
 - ▶ Le type de fichier (fichier standard, lien symbolique, répertoire, ...)
 - ▶ Les droits d'accès
 - ▶ le propriétaire (UID)
 - ▶ le groupe propriétaire (GID)
 - ▶ le nombre de références
 - ▶ la taille
 - ▶ la date de dernier accès en écriture
 - ▶ la date de dernier accès en lecture
 - ▶ l'adresse disque du premier bloc de données
- Toutes les informations fournies par **ls -l** s'y trouvent, sauf **le nom**.
- La racine du système de fichier (/) correspond à l'inode numéro 2.

Les répertoires

- Un répertoire est simplement un fichier, dont
 - ▶ le type est différent
 - ▶ les blocs de données contiennent une liste de couples (**nom, numéro d'inode**)
- Si le répertoire contient 3 fichiers *a.txt*, *b.sh*, *c.pl*, il sera composé d'une liste de 5 couples :
 $\{ (., 43), (., 234), (a.txt, 36), (b.sh, 245), (c.pl, 24524) \}$

Liens: Lien physique

Un lien est un couple (**nom-de-fichier, numéro d'inode**). On peut créer 2 types de liens :

- Les liens *physiques* :
 - ▶ création d'un nouveau nom pour le même inode.
 - ▶ impossible de faire un lien physique vers un répertoire : le système de fichiers doit être arborescent.
 - ▶ impossible de faire un lien physique vers un autre volume (partition)

Lien symbolique

- Les liens *symboliques* (raccourcis) :
 - ▶ création d'un nouvel inode de type *lien symbolique*
 - ▶ le bloc de données contient un chaîne de caractères représentant le chemin (relatif ou absolu) vers un fichier.
 - ▶ possibilité de lien symbolique vers un répertoire.
 - ▶ possibilité de lien symbolique vers un autre volume (partition)

Commandes utiles

- - ▶ Pour créer un lien physique : **ln cible.txt nomDuLien.txt**
 - ▶ Pour créer un lien symbolique : **ln -s cible.txt nomDuLien.txt**

ls -li permet d'afficher, pour chaque fichier, le numéro d'inode.

ls -ali permet de voir en plus les fichiers cachés (. et .. en particulier).

Scripts

Scripts Shell

- L'interprète de commandes (shell) permet d'interagir avec le système
- Exécution de commandes (modification / consultation de l'état du système)
- Utilisation avancée : combinaison de commandes par exemple en redirigeant les entrées/sorties
- Programmation shell : combinaison de commandes au sein d'un script dans le but d'automatiser certaines tâches
- Un Script shell correspond à un fichier exécutable d'extension .sh et débutant par : `#!/bin/sh`

Commentaires

- Les commentaires dans les scripts sont introduits par le symbole #.

Shell : variables

Les variables

Association entre un nom et un contenu (chaîne de caractères, nombre entier)

Affectation d'une valeur à une variable au moyen du symbole =

Exemple: `$ x="ceci est une variable"`

Lecture de la valeur d'une variable au moyen du symbole \$

Exemples:

```
$ echo $x
```

```
$ echo "oui, "$x
```

NB: les noms de variables peuvent être des entiers, dans ce cas, ils doivent être entourés de {} à partir de 10 :

```
$ echo ${10}
```

Attention : une variable non-définie contient la chaîne vide!

Les variables

- Possibilité de passer des paramètres à un script shell au moyen d'arguments.
- Les arguments sont définis dans la ligne de commande, à la suite du nom de l'exécutable, et séparés par des espaces.
- Dans le script, le nom de l'exécutable est associé à la variable \$0, les arguments aux variables \$1, \$2, . . .
- La variable \$# contient le nombre entier d'arguments du script.
- La variable \$* contient la concaténation de tous les arguments.

Utilisation des variables

- nom de variable : HOME
- valeur de la variable : \$HOME
- afficher la valeur de la variable : **echo \$HOME**
- créer et affecter une variable : **VAR=toto**
- supprimer une variable : **unset VAR**
Attention : pas d'espace autour de l'opérateur "="

Le PATH

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
```

- Cette variable contient une *liste de chemins de répertoires* dans lesquelles le Shell ira chercher les exécutables.
- Si un exécutable donné ne se trouve pas dans le PATH, le Shell ne pourra pas la trouver, et nous donnera un message d'erreur :
bash: cmd: command not found

Affectation des variables

- créer une variable avec l'affectation : **VAR=toto**
Attention : pas d'espace autour de l'opérateur "="
- commande **read variables...** affecte à chaque *variable* un mot de la première ligne lue sur l'entrée standard.

```
read var1 var2 var3 var4
```

Fonctionnement :

- 1 **read** attend qu'une ligne arrive sur l'entrée standard (`\n`).
- 2 Cette ligne est découpée en mots (séparateurs : espace, tabulation).
- 3 Le *nⁱeme* mot est affecté à la *nⁱeme* variable.
- 4 S'il y a plus de mots que de variables, la dernière variable contient toute la fin de la ligne.

arguments génériques

- `$#` est équivalent à `argc (-1)` en C : c'est le nombre d'arguments que l'on a passé au script lors de l'appel.
- `"$@"` est équivalent à `argv` en C : c'est la liste des arguments que l'on a passé au script.

accès direct

- \$0 est le nom complet du script (pour avoir le nom court : **basename \$0**)
- \$1 est le premier argument passé au script.
- ...
- \$9 est le neuvième.

Impossible d'accéder directement aux suivants (\$10 n'existe pas).

Expressions arithmétiques

- Le shell peut évaluer des expressions arithmétiques délimitées par `$(( ))`

Exemple:

```
n=1
```

```
echo $(( n + 1 ))
```

```
p = $(( n * 5 / 2 ))
```

```
echo $p
```

Evaluation de commandes

- Possibilité de stocker le résultat d'une commande dans une variable.
- Utilisation de la backquote (`)
- Exemple :
 `n=`ls | wc -l``
 `echo $n`

Erreur de commande

Si la commande est erronée, un message d'erreur apparaît :

```
tichit@iml230:~$ heure
```

```
bash: heure : commande introuvable
```

Les messages d'erreur apparaissent sur la **sortie standard d'erreur**. Cette sortie est redirigée sur le terminal courant.

```
tichit@iml230:~$ rep -
```

```
bash: rep: commande introuvable
```

```
tichit@iml230:~$ cat toto
```

```
cat: toto: Aucun fichier ou répertoire de ce type
```

```
tichit@iml230:~$ rm .
```

```
rm: ne peut enlever « . » or « .. »
```

```
tichit@iml230:~$ rm public_html/
```

```
rm: ne peut enlever 'public\_html/': est un répertoire
```

Le code de retour

Il s'agit d'un **entier** permettant de savoir si une commande a réussi ou échoué.

- Déroulement normal : statut 0.
- Déroulement anormal : statut différent de 0.

Le code de retour sera rendu dans la variable “?”, ce qui permettra de réaliser des opérations conditionnelles lors de l'écriture de **scripts** shell.

```
tichit@iml230:~$ whoami
```

```
tichit
```

```
tichit@iml230:~$ echo $?
```

```
0
```

```
tichit@iml230:~$ rm public_html/
```

```
rm: ne peut enlever 'public\_html/': est un répertoire
```

```
tichit@iml230:~$ echo $?
```

```
1
```

La commande exit

Tout script doit indiquer au système s'il s'est déroulé avec succès ou s'est terminé sur un échec.

Pour cela, on se sert de la commande **exit entier** qui termine l'exécution du script.

exit 0 indique un succès, **exit n** (avec $n \neq 0$) indique un échec.

structures de contrôle

Commande if

Syntaxe 1 :

```
if [ condition ]  
then  
    action1  
fi
```

Syntaxe 2 :

```
if [ condition ]  
then  
    action1  
else  
    action2  
fi
```

Commande if

Les blocs `elif` et `else` sont facultatifs.
commande1 peut réussir ou échouer.
Si elle réussit, le bloc *then* est effectué.
Si elle échoue, le bloc *else* est effectué.

Commande if

- Exemples :

```
if [ $# -eq 0 ]  
then  
    echo "$0 : Aucun argument reçu !"  
fi
```

```
if test `cp "$1" "$1%"`  
then  
    echo "sauvegarde de $1 reussie"  
else  
    echo "sauvegarde du fichier $1 impossible"  
fi
```

Instructions if imbriquées

```
if [ condition 1 ]  
then  
    action1  
elif [ condition 2 ]  
then  
    action2  
elif [ condition 3 ]  
    action3  
...  
else  
    actionN  
fi
```

Commande test

La commande **test** est une commande qui ne fait rien d'autre que réussir ou échouer.

Elle sert très souvent d'argument à la commande **if**.

Elle possède un grand nombre d'options.

Elle peut aussi s'écrire [...]

Ex :

- **test -f /home/test** ou [**-f /home/test**] permet de savoir si le fichier */home/test* est un fichier ordinaire.
- **test \$1 \> a** permet de savoir si le premier argument est supérieur (lexicographiquement) à "a".
- **test \$1 = help** permet de savoir si le premier argument est "help".
- **test \$# -gt 3** permet de savoir si le nombre d'arguments est supérieur ou égal (numériquement) à 3.
- ...

Boucle while

```
while commande1 ; do  
  commande2  
  commande3  
done
```

Si on redirige l'entrée standard du **while** (ou plutôt du **done**) vers un fichier, **commande1** aura aussi son entrée standard redirigée vers ce fichier.

Boucle for

```
for variable in liste ; do  
  commande1  
  commande2  
done
```

variable est un nom de variable libre. *liste* est en fait une chaîne de caractères qui devrait contenir des caractères de séparation (tabulations, espaces).

Boucle for sans arguments

```
for variable ; do  
  commande1  
  commande2  
done
```

idem à

```
for variable in "$@" ; do  
  commande1  
  commande2  
done
```

Exemple

- Exécutez le script suivant :

```
x=1
```

```
while [ $x -le 5 ]
```

```
do
```

```
echo "Hello World $x"
```

```
x=$(( $x + 1 ))
```

```
done
```

Exemple

- Exécutez le script suivant :

```
counter=$1
```

```
factorial=1
```

```
while test $counter -gt 0
```

```
do
```

```
factorial=$((factorial * counter ))
```

```
counter=$((counter - 1 ))
```

```
done
```

```
echo $factorial
```

Exemple

- Exécutez le script suivant :

```
while :
```

```
do
```

```
echo "boucle infinie [ CTRL-C pour arreter ]"
```

```
done
```

Exemple

- Exécutez le script suivant :

```
for i in 1 2 3 4 5
```

```
do
```

```
echo "Hello World $i"
```

```
done
```

Exemple

- Exécutez le script suivant :

```
for i in {1..5}
```

```
do
```

```
echo "Hello World $i"
```

```
done
```

Exemple

- Exécutez le script suivant :

```
for (( c=1; c<=5; c++ ))  
do  
echo "Hello World $c"  
done
```

Exemple

- Exécutez le script suivant :

```
for (( ; ; ))
```

```
do
```

```
echo "boucle infinie [ CTRL-C pour arreter ]"
```

```
done
```

switch

```
case variable in
  expReg1)
    cmd1
    cmd2
    ;;
  expReg2)
    cmd3
    cmd4
    ;;
  *)
    cmd5
    cmd6
esac
```

- variable est un nom de variable libre.
- expReg1 et expReg2 sont des expressions régulières.

Exemple

```
case $# in
o)
  echo "aucun parametre"
  echo "Syntaxe : $0 <nom d'utilisateur> "
;;
1)
  echo "1 parametre passe au programme : $1"
  ;;
2)
  echo "2 parametres passes au programme : $1 et $2«
  ;;
*)
  echo "TROP DE PARAMETRES !"
esac
```

Opérateurs de comparaison

- Tests sur les fichiers (et sur les répertoires) :
 - **-e fichier** Vrai si le fichier/répertoire existe
 - **-s fichier** Vrai si le fichier à une taille supérieure à 0
 - **-z fichier** Vrai si le fichier fait 0 octet (donc si il est vide)
 - **-r fichier** Vrai si le fichier/répertoire est lisible
 - **-w fichier** Vrai si le fichier/répertoire est modifiable
 - **-x fichier** Vrai si le fichier est exécutable ou si le répertoire est accessible
 - **-O fichier** Vrai si le fichier/répertoire appartient à l'utilisateur
 - **-G fichier** Vrai si le fichier/répertoire appartient au groupe de l'utilisateur

Opérateurs de comparaison

- Tests sur les fichiers (et sur les répertoires, suite) :
 - **-d nom** Vrai si nom représente un répertoire
 - **-f nom** Vrai si nom représente un fichier
 - **-L nom** Vrai si nom représente un lien symbolique
 - **f1 -nt f2** Vrai si les deux fichiers existent et si f1 est plus récent que f2
 - **f1 -ot f2** Vrai si les deux fichiers existent et si f1 est plus ancien que f2
 - **f1 -ef f2** Vrai si les deux fichiers représentent un seul et même fichier

Opérateurs de comparaison

- Tests sur les entiers :
 - **entier1 -eq entier2** Vrai si entier1 est égal à entier2
 - **entier1 -ge entier2** Vrai si entier1 est supérieur ou égal à entier2
 - **entier1 -gt entier2** Vrai si entier1 est strictement supérieur à entier2
 - **entier1 -le entier2** Vrai si entier1 est inférieur ou égal à entier2
 - **entier1 -lt entier2** Vrai si entier1 est strictement inférieur à entier2
 - **entier1 -ne entier2** Vrai si entier1 est différent de entier2

Opérateurs de comparaison

- Tests sur les chaînes de caractères :
 - `-n "chaîne"` Vrai si la chaîne n'est pas vide
 - `-z "chaîne"` Vrai si la chaîne est vide
 - `"chaine1" = "chaine2"` Vrai si les deux chaînes sont identiques
 - `"chaine1" != "chaine2"` Vrai si les deux chaînes sont différentes