Systèmes d'Exploitation

Valeur d'accueil et reconversion en informatique 1 (VARI1)

Daniel Porumbel

nombreaux slides dus à

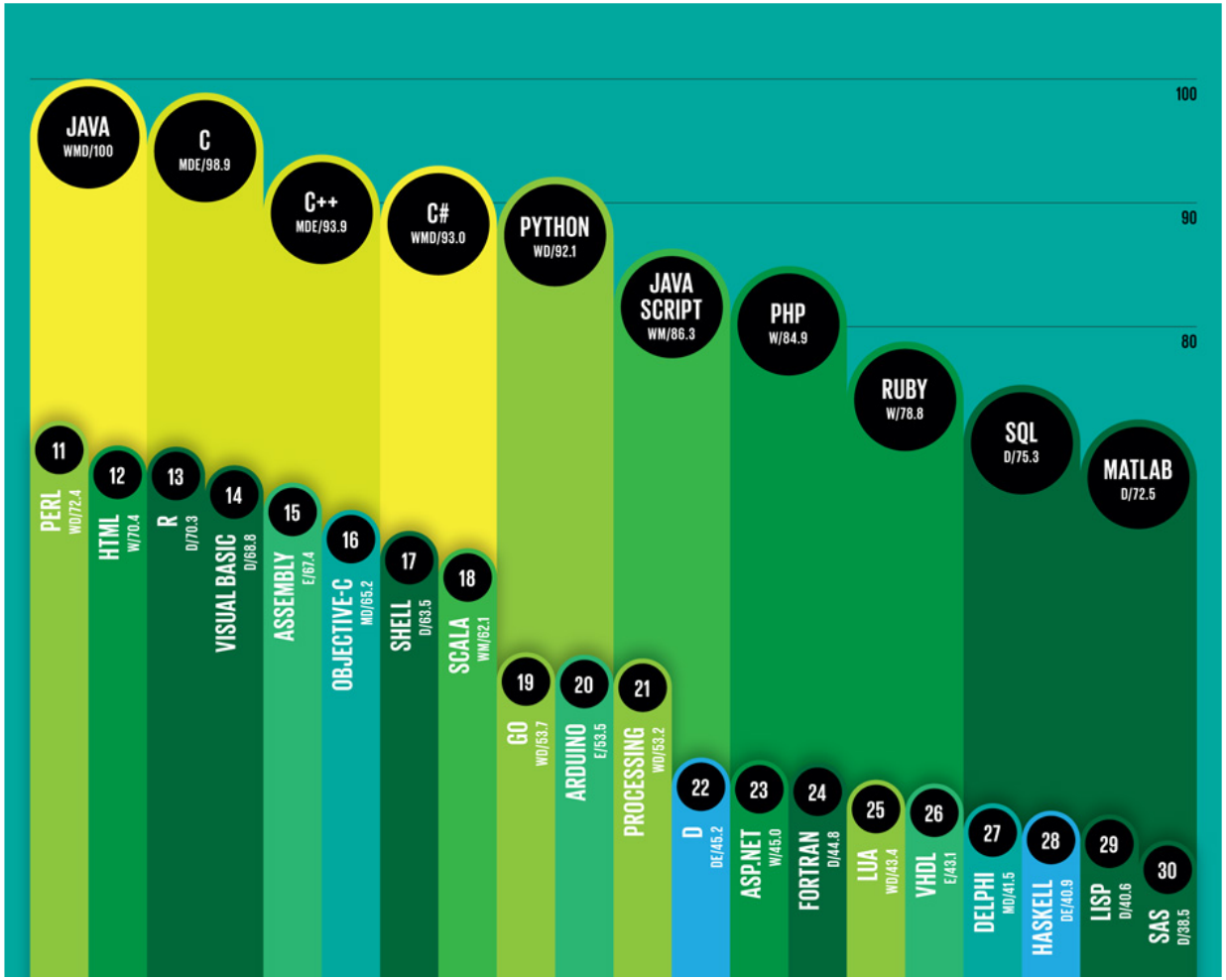
Pierre Cubaud, Bertrand Le cun, Emmanuel Hyon

1. Notions de Programmation

2. Systèmes d'Exploitation

Plan

Popularité des langages (d'apres IEEE Spectrum, sept 2014)



base 100 pour le plus populaire (ici, Java)

Identifiants autorisés en Java/Processing

- sensible à la casse : CASSE≠CaSsE
- on peut utiliser le $ et le _ (underscore)
- on peut utiliser des accents mais ce n'est pas conseillé
  - risque de problème de codage

Coutumes avec la casse

- *data* : une variable, un paquetage
- *Hello* : une classe
- *lireSuivant* : une "grosse" variable
- *MAXWIDTH* : une constante

Mauvais style

- *ceciestunnomtroplongdevariable*

La manipulation de chaines

```
1 String chaine1 = "ab";              //chaine1="ab"
2 String chaine2 = chaine1+"cd";//chaine2="abcd"
3 int i=2,j=3;
4 String chaine3 = "abc"+i*j;     //chaine3="abc6"
5 String chaine4 = "abc"+i+j;     //chaine4="abc23"
6 String chaine5 = ""+i+j;        //chaine5="23"
7 String chaine6 = i+j;           //erreur de con-
8                                 //version int->String
```

Mots reservés de Java

abstract	continue	for	new	switch
assert[Note 1]	default	goto[Note 2]	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum[Note 3]	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp[Note 4]	volatile
const[Note 2]	float	native	super	while

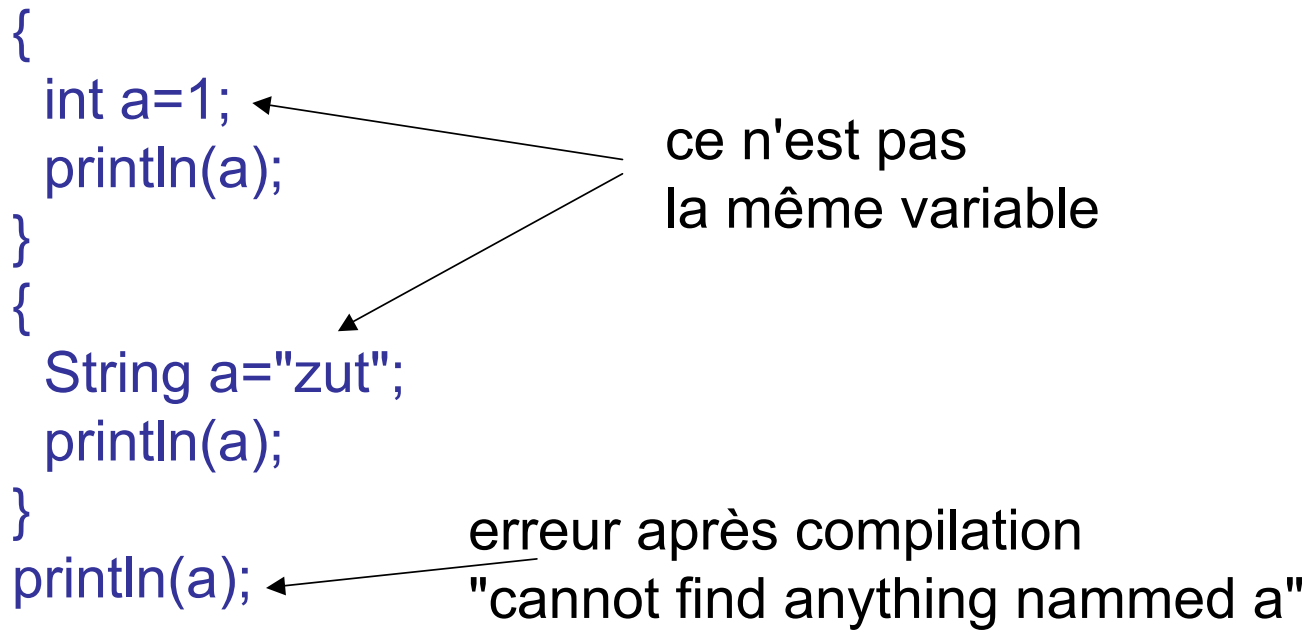
1. ^ Keyword was introduced in J2SE 1.4
2. ^ [a] [b] Keyword is not used
3. ^ Keyword was introduced in J2SE 5.0
4. ^ Keyword was introduced in J2SE 1.2

(wikipedia)

Rque : attention aussi à ceux de Processing

Blocs et portées des déclarations

- les blocs de traitements sont délimités par un { et un }
- ils peuvent être imbriqués arbitrairement
- les variables déclarées au sein d'un bloc sont locales à ce bloc



Blocs imbriqués :

```
{
  int a=1;
  {
   String a="zut";
  }
println(a);
}
```

illégal : le bloc contenu a accès aux variables du bloc contenant

Coutume concernant l'indentation :
on décale chaque niveau par des espaces pour rendre le code plus lisible

Intérêt des variables locales : occupation mémoire limitée au moment du traitement utile

Plan

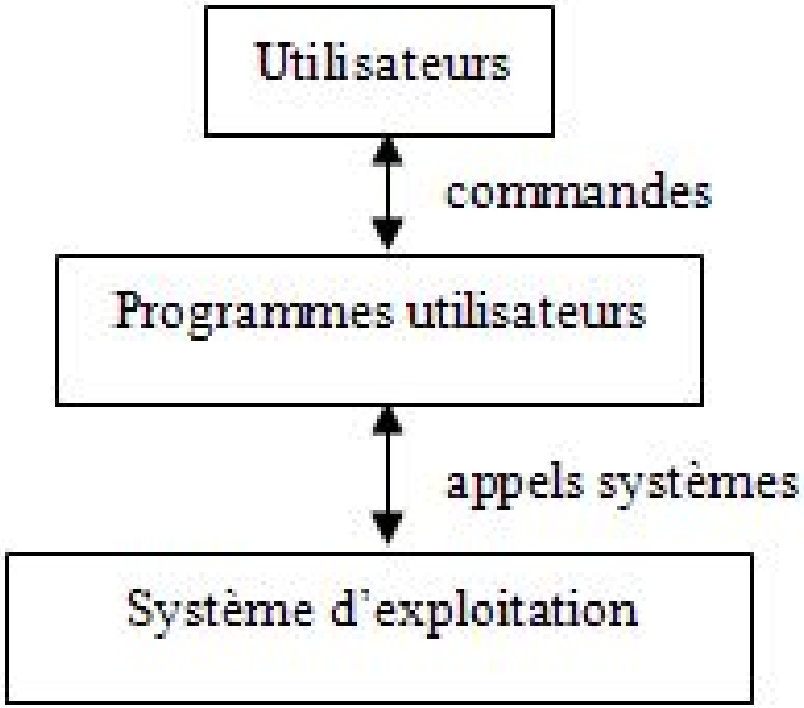
Système d'exploitation : définition

Un système d'exploitation

un ensemble de programmes qui dirige l'utilisation des capacités d'un ordinateur par des applications

- le système d'exploitation est le premier programme exécuté lors de la mise en marche de l'ordinateur

- Au tout début, les machines ne possédaient pas de système d'exploitation.
- Pour les faire fonctionner, il fallait que les programmes utilisateurs gèrent directement les périphériques ce qui impliquait une connaissance fine des caractéristiques physiques et des spécificités de chaque composant.
- Pour libérer les programmeurs de la gestion du matériel, on a introduit une couche logicielle sous la forme d'un ensemble de sous-programmes (appels système) standard,



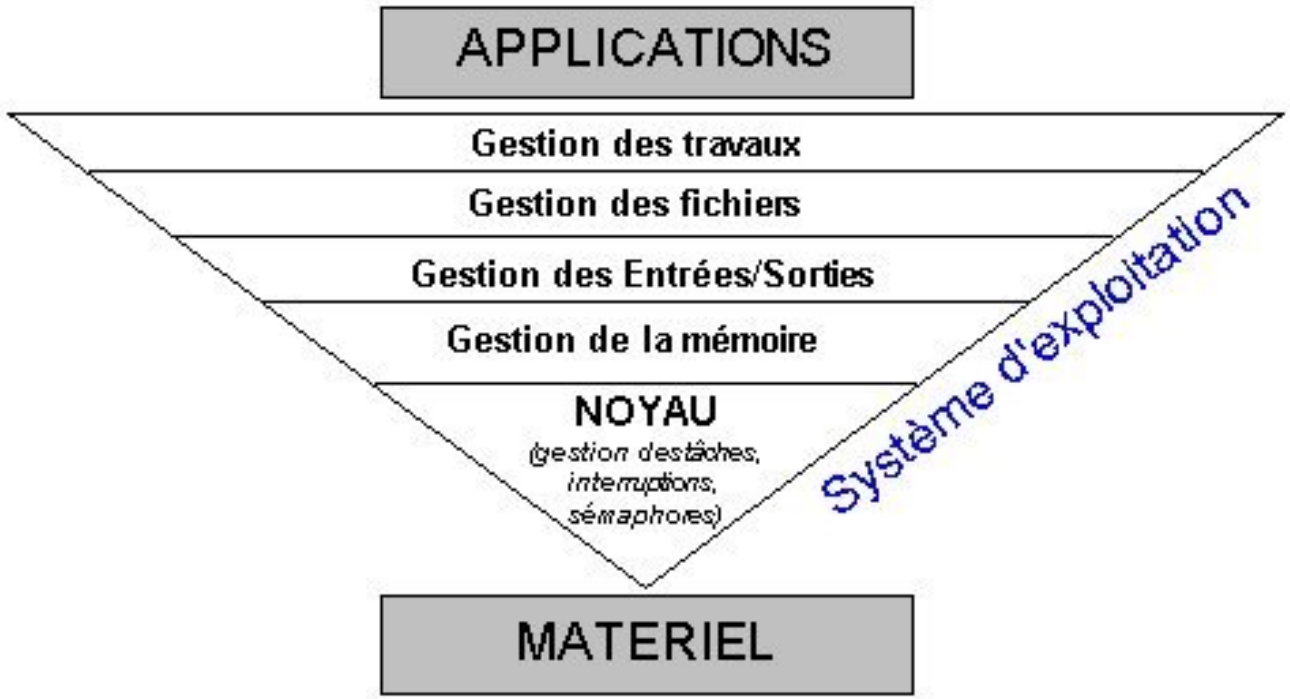
Système d'exploitation : machine « abstraite » **indépendante** du matériel

Langage Humain $\rightarrow$ Langage du Matériel

Le système d'exploitation doit gérer l'exécution de la chaîne :
action utilisateur$\rightarrow$*programmes*$\rightarrow$*action physique*

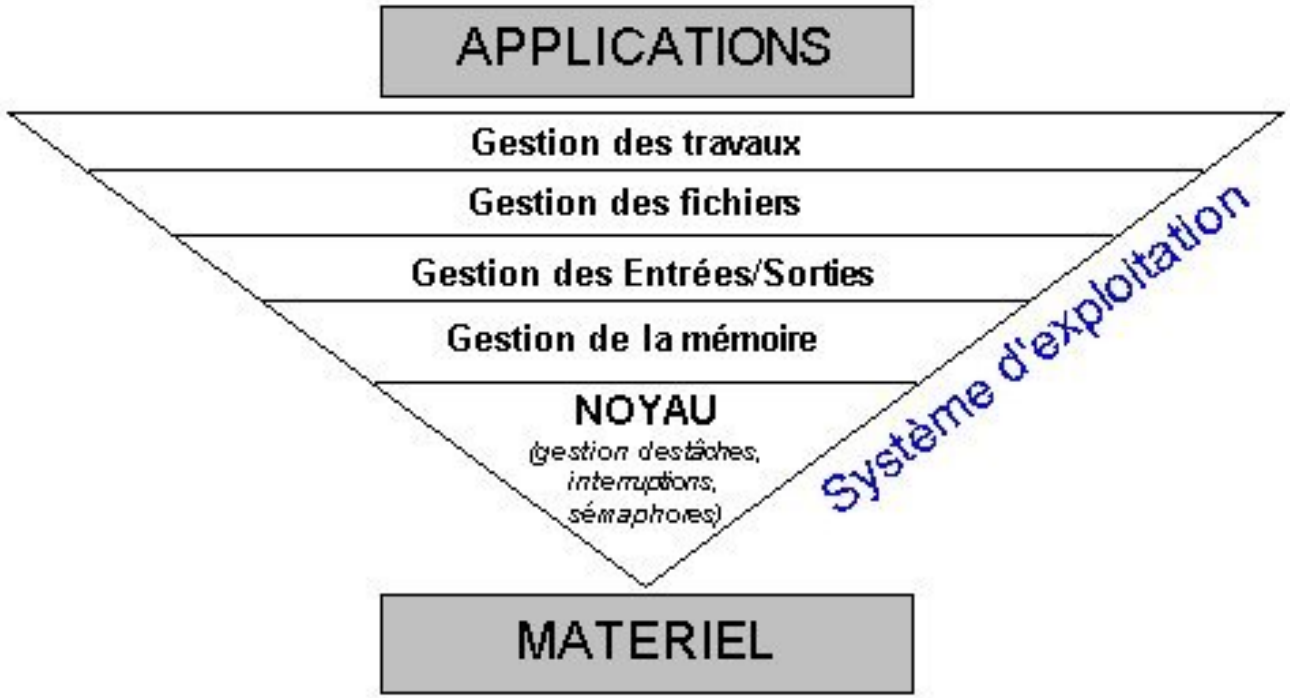
Langage Humain → Langage du Matériel

Le système d'exploitation doit gérer l'exécution de la chaîne :
action utilisateur→programmes→action physique



Langage Humain → Langage du Matériel

Le système d'exploitation doit gérer l'exécution de la chaîne :
action utilisateur→programmes→action physique



Composition

gestion des processus, communication inter-processus, la gestion de la mémoire, sécurité des données et communications réseau, systèmes de fichiers, pilotes, utilisation processeur, interface utilisateur, programmes utilitaires, etc.

Taches Importantes Gérés par le SE

- la communication avec le clavier, l'écran, le disque dur, les périphériques
  - Linux : tout périphérique est vu comme un fichier par les programmes
- l'exécution des programmes et l'accès en parallèle à la mémoire
- la gestion des pilotes (anglais : drivers)
  - pilote=ensemble d'instructions à exécuter pour utiliser un certain périphérique informatique (ex., l'imprimante).

Les systèmes

Système Propriétaire : Windows, Apple (*Darwin* Free mais *aqua* et *quartz* non free)

- beaucoup d'aspects sont secrets
- les changements et évolutions sont imposés
- fonctionnalités caché

Systèmes Ouverts (Unix, Linux)

- logiciels libres
- tout est public
- les changements et évolutions sont discutés et votés

Interfaces utilisateur

Tout système propose des possibilités pour manipuler les fichiers et les dossiers et pour lancer des programmes

- Sous MacOS, Windows, tout à la souris
  - click-click sur des icones
  - click sur des menus déroulants
- Sous Linux, à l'origine tout au clavier
  - interpréteur de commandes (shell, terminal ou console)
  - beaucoup des commandes disponibles (cs, ls, rm, cp, mv, rm, etc) ;

Communication avec le clavier et l'écran

Principes LINUX

- le clavier = le fichier `/dev/tty`
- le terminal = le fichier indiqué par la commande `tty`
- l'écran est géré par un serveur *X*.
  - Les programmes sont des clients qui envoient des requêtes au serveur *X*
  - La commande `ssh -X IPDIST` permet de lancer sur l'écran local des programmes qui tournent sur `IPDIST`

Le Shell

Principes

- programme qui attend que vous tapiez des commandes,
- le prompt ou “invite ” indique cette attente.
- Interpréteur ligne une commande sur une ligne, en appuyant sur entrée (caractère fin de ligne) la commande sera exécutée,
- Beaucoup de petites choses facilitent l’utilisation du shell.
- Propose aussi un véritable langage de programmation les scripts

Le Shell bases

Notion de répertoire courant

- Comme tout processus unix, le shell a un répertoire courant.
- C'est à partir de ce répertoire que ce fait le nommage des fichiers ou répertoires.

Système de nommage

- répertoire courant "."
- répertoire père ".."
- séparateur de répertoire "/"
- répertoire HOME "~"
- répertoire racine (root), on commence par "/"

Système de nommage exemple

Référence absolue

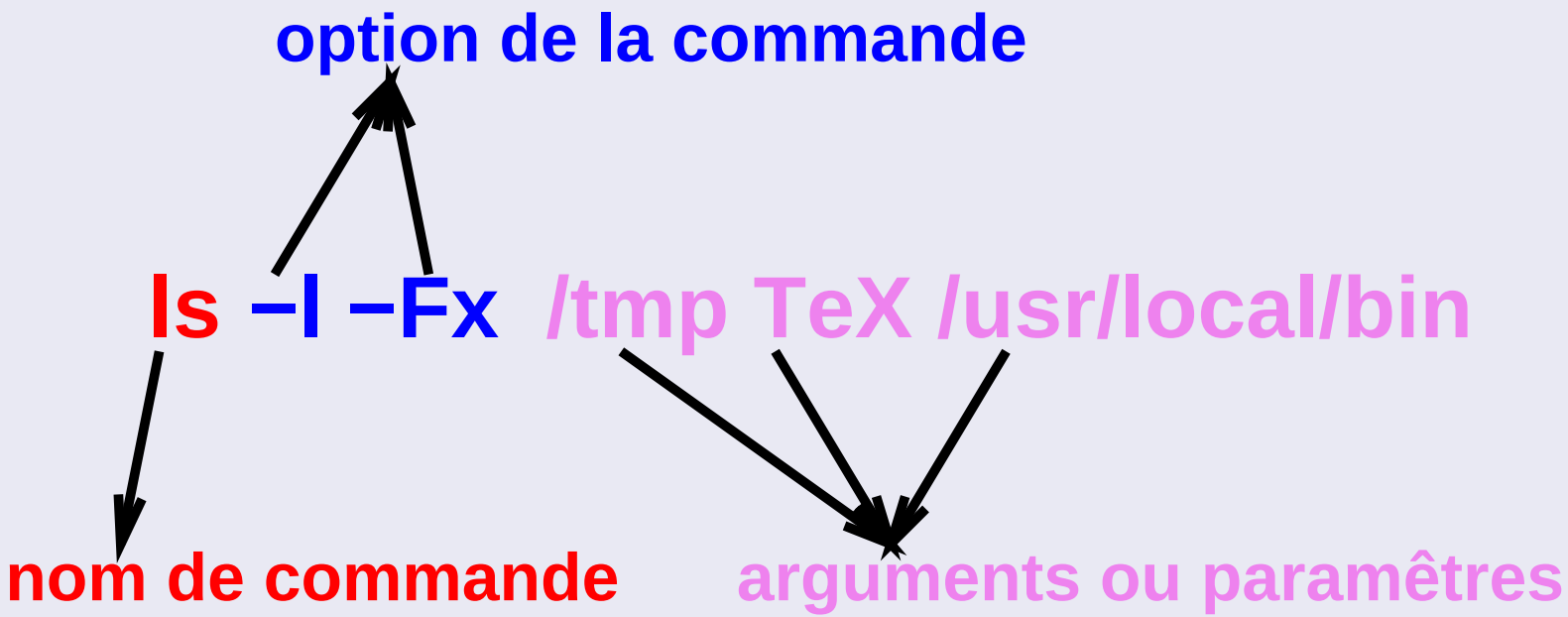
- donnée à partir de la racine :
  Exemple : `cd /home/users/info/moi/TeX/Publi`
- donnée à partir d'un HOME :
  Exemple : `ls ~/TeX/Publi` ou `ls ~moi/TeX/Publi`

Référence relative

- donnée à partir du répertoire courant.
  `cd TeX/Publi` ou `cd ./TeX/Publi`
  `cd ../user/TeX`
  `cd ../../toto/Work/latex`

Forme générale d'une commande sous unix

Commande ligne



une commande est sur une ligne.

Les commandes

Commandes usuelles

- `cd répertoire` change le répertoire courant du shell,
- `cd` ou pwd affiche le rpertoire courant du shell,
- `ls` ou ls <liste de fichiers ou catalogue> affiche le nom des fichiers et catalogues du répertoire courant.
- `cp f1 f2` copie du fichier f1 dans f2.
- `cp f1 f2 f3 ... d1` copie des fichiers f1 f2 f3 ... dans le catalogue d1.

Les commandes

Commandes usuelles

- `rm f1 f2 f3` suppression des fichiers f1 f2 f3
- `mv f1 f3` changement de nom du fichier f1 en f2.
- `mv f1 f2 f3 ... d1` changement de location des fichier f1 f2 f3 ... dans le répertoire d1.
- `mkdir d1` crée le répertoire d1.
- `rmdir d1` détruit le répertoire d1 (il doit être vide).
- `file f1 f2 f3` affiche le type des fichiers f1 f2 f3.
- `cat f1 f2 f3` affiche le contenu des fichiers f1 f2 f3.
- `more f1 f2 f3` affiche page à page le contenu des fichiers f1 f2 f3.

Les commandes et leur manuel

la commande man

- `man <commande>` donne le manuel de la commande en paramètre
- `man -k <mot>` donne la liste des pages man contenant le <mot>
- Les pages de man sont classées par section.
  - Section 1 : commandes utilisateurs
  - Section 2 : appels systèmes
  - Section 3 : fonctions de bibliothèques
  - Section 4 : *devices*
  - Section 5 : format de fichiers spécifiques
  - Section 6 : jeux
  - Section 7 : divers
  - Section 8 : commandes d'administration système

Par exemple, getopt existe dans 1 et 3

man 1 getopt

man 3 getopt

Expansion des noms de fichiers

Nommer plusieurs fichiers de noms différents

ls *.tex Affiche les fichiers du répertoire courant ayant l'extension '.tex'.

ls T*.tex Affiche les fichiers du répertoire courant commençant par 'T' et ayant l'extension '.tex'.

ls ?a*b*.tex Affiche les fichiers du répertoire courant ayant pour deuxième lettre un 'a', contenant un 'b' et ayant l'extension '.tex'.

ls ? ? ?.tex Affiche les fichiers du répertoire courant de trois caractères et ayant l'extension '.tex'.

D'autres sont aussi possibles voir le man du shell correspondant

- Attention le shell effectue les expansions de noms de fichier, pas la commande.

Compléter un nom de fichier

Complétion des noms de fichiers

- Expansion automatique des noms de fichiers
  Touche [TAB] étend automatiquement les noms de fichiers (et commandes) correspondant aux caractères tapés.

```
<Mo-lecun-4-68- Unix -> ls
Net.F        disk.F       main.dvi    main.tex
OS.F         display.F    main.log    psfig.tex
cmdfrm.F     main.aux     main.ps
<Mo-lecun-4-69- Unix -> ls ma<TAB>
main.aux  main.dvi  main.log  main.ps   main.tex
<Mo-lecun-4-69- Unix -> ls main.
```

Voir les commandes déjà exécutées

Historique des commandes

- le shell maintient une liste des commandes que vous avez tapées.
- les flèches haut et bas permettent de naviguer dans cet historique.
- commande “history” affiche cette liste.
- commande “!ma” ré-exécute la dernière commande dont le nom commence par ma
- commande “!23” ré-exécute la commande numéro 23
- commande “!-1” ré-exécute l’avant dernière commande

Modes d'exécution des commandes

Avant-plan

- Lorsqu'une commande est lancée par le shell, elle prend "la main" sur le shell.
- Le shell ne peut plus lire d'autres commandes au clavier
- Le shell est en attente que la commande se termine

Arrière-plan

- Possibilité de lancer une commande en arrière plan
- Ajouter un & à la fin de la commande
- Juste après avoir lancé la commande, le shell attend de nouvelles commandes

Redirections

Entrée et sortie et sortie erreur standard

- Toute application a
  - une entrée standard : pour lire des données
  - une sortie standard : pour afficher des données
  - une sortie erreur standard : pour afficher des erreurs d'exécution
- généralement ces entrées/sorties standards sont le terminal actif

Rediriger les E/S d'une application

- Ajouter `> fic` à la fin d'une commande permet de rediriger les affichages de la commande dans le fichier fic
- Ajouter `< fic` à la fin d'une commande permet de rediriger l'entrée standard de la commande à partir du fichier fic. La commande lira ces données dans ce fichier (et non au clavier).
- Ajouter `>& fic` à la fin d'une commande permet de rediriger les affichages d'erreurs de la commande dans le fichier fic

Le pipe

le pipe

- permet de rediriger la sortie d'une commande vers l'entrée d'une autre.
- Exemples :
  - `ls | wc -l` : compte le nombre de fichier du répertoire
  - `ls -l | more` : permet de voir les fichiers du dossier courant de manière paginée
  - `ls /bin /usr/bin /usr/local/bin | grep X | sort | more` : permet de voir de façon triée et paginée les fichiers contenus dans les dossiers /bin /usr/bin /usr/local/bin contenant un X dans le nom.

Composition de commandes

Séquence de commandes

- pour un shell : une ligne une commande
- ; composition : le caractère ; permet de mettre plusieurs commandes sur une même ligne.

```
cd ; ls -l
```

Composition conditionnelle

|| ou &&

sont les exécutions conditionnelles

- && : exécuter une commande que si la première est vraie,
- || : ne pas exécuter une commande si la première est vraie.

exemple : `cd /var/log/mail && ls -l`

exemple : `cd /var/log/mail || cd /var/mail`

Tout programme C est une “commande”

Tout programme exécutable (le binaire) peut être vu comme une commande. En effet la syntaxe du lancement du programme exécutable est similaire à une commande.

exemple : `./monprograme`

ou

`monprogramme`

Tout programme C est une commande (2)
Interaction avec environnement

Passage des arguments

Prototype :

```
main(int argc, char *argv[])
```

- `argc` est le nombre d'arguments (nom du programme compris),
- `*argv[]` est un tableau de chaînes de caractères :
  - `argv[0]` : adresse d'une chaîne contenant le nom du programme,
  - `argv[1]` : adresse du mot suivant.

Exemple

```
int main(int argc, char *argv[]) {

if (argc > 1) printf("Bonjour %s",argv[1]);
}
```

Groupement de commandes

Utilisation des ‘ ‘

Récupérer le résultat d’une commande

- on peut exécuter une commande et récupérer son résultat dans une variable

```
:~:> set f=‘ls  | grep t‘
:~> echo $f
CourSystRes.tgz intro.aux intro.tex main.out main.tex
:~>
```

Attention il s’agit du signe ‘ (touche AltGr 7).

Les Alias

Mettre sous un mot un groupe de commandes

- bash : `alias ll='ls -l'`
  Pas d’alias en sh, ksh et bash ont les alias mais sans paramètres.

Instructions des shells

Véritable langage de programmation

- Les shells sont des véritables langages de programmation mais orientés fichiers
- Généralement, une suite de commandes mise dans un fichier représentant un script
- Facilitent
  - lecture d'un répertoire
  - lecture des attributs de fichiers (droits, type, etc)
  - permet de lancer des commandes, programmes

Les fichiers scripts

Forme du fichier

- un fichier script est un fichier texte (ascii), une suite de caractères.
- la première ligne contient le binaire permettant d'interpréter le fichier.
- pour sh

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

- il existe sous unix d'autres langages de script : Perl, Tcl, Ruby...
- Équivalent du .bat sous windows

Lancement

- Le nom du script est une commande (si fichier est executable).
- Sinon c'est un paramètre de la commande `sh`

Les arguments des fichiers scripts

Des variables stockent les arguments du script

- $0 le nom du script $1 $2 $3 respectivement le premier, deuxième troisième argument. $* tous les arguments
- $# le nombre de paramêtres

```
:~$ cat truc.sh
#! /bin/sh
echo nombre de parametre $#
echo le script est $0
echo premier argument est $1
:~$ sh truc.sh titi
nombre de parametre 1
le script est truc.sh
premier argument est titi
:~$
```