Introduction à l'Informatique

Le système Unix

Cours 1

kn@lri.fr

université PARIS-SACLAY

La science informatique

Informatique (*computer science*): ensemble des sciences qui ont pour objet d'étude **l'information** et les **procédés** de traîtement **automatique** de celle-ci.
Quelques exemples de champs d'étude couverts par cette définition :

- L'architecture matérielle des machines
- Les systèmes d'exploitation
- Les réseaux informatiques
- Les systèmes de gestion de bases de données
- Les langages de programmation
- Le génie logiciel
- L'algorithmique
- La complexité et la calculabilité
- La logique
- Les interfaces et l'interaction Humains-Machines
- ...

Plan

1 Présentation du cours ✓
2 Le système Unix
2.1 Principes des systèmes d'exploitation
2.2 Système de gestion de fichiers

Système d'exploitation

Quelques systèmes:

- Windows XP/NT/2003/7/8/10/11, ...
- Linux, FreeBSD, NetBSD, OpenBSD, ...
- MacOS X (basé sur une variante de FreeBSD), ...
- Unix, AIX, Solaris, HP-UX, ...
- iOS, Android, ...

Programme

Un *programme* est une suite *d'instructions* réalisant une certaine *tâche* sur un *ordinateur*.

Exemple de programmes :

- Le navigateur Web Firefox est un programme
- DOTA (jeux vidéo) est un programme
- La partie logicielle du système de navigation d'une voiture est un programme
- La suite d'instructions qui fait vibrer le téléphone quand on reçoit un message est un programme.

Parfois, un programme est appelé *app* (diminutif de *application*), ~~de façon complètement ridicule~~, à des fins de marketing.

Système d'exploitation

Qu'est-ce qu'un système d'exploitation ?

- c'est un *programme*
- qui *organise* l'accès aux *ressources* de la machine

Quelles sont les ressources d'une machine?

- Processeur (temps d'exécution)
- Mémoire
- Accès aux périphériques de stockage
- Accès aux périphériques d'entrées/sorties
- ...

Système d'exploitation

Haut niveau — *Applications (programmes utilisateurs)*: navigateur Web, éditeur de texte, anti-virus, jeu, compilateur, ...

↓

Système d'exploitation:

- Gestion des ressources
- Interface avec le matériel (pilotes)

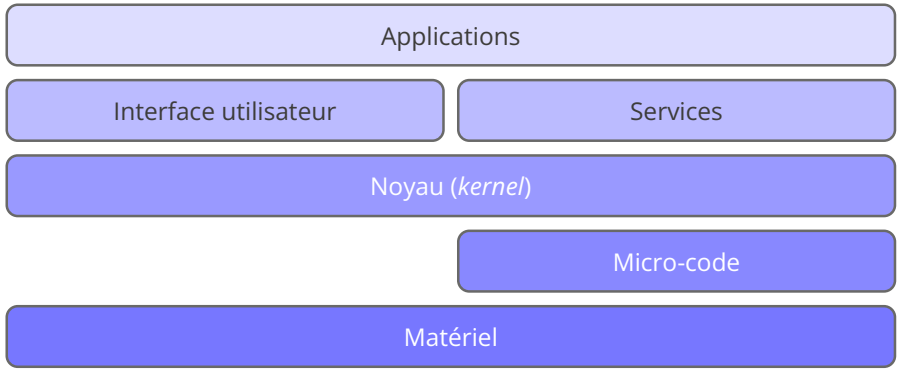
Bas niveau — *Matériel*: processeur, mémoire, périphériques, ...

Le système Unix

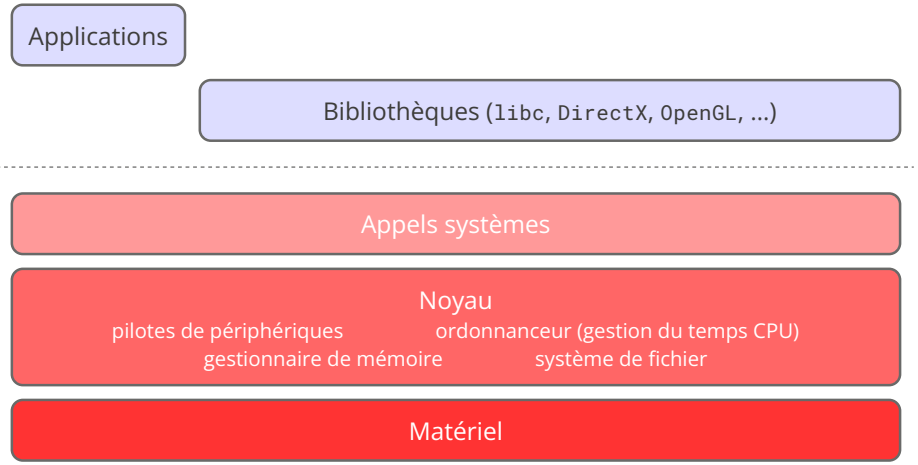
1965 :	MultICS: *Multiplexed Information and Computing Service* (Bell & MIT)
1969 :	Unix: 1ère version en assembleur (AT&T)
1972-73 :	Unix réécrit en C
1976 :	Invention de TCP/IP
1977 :	*Berkeley Software Distribution* (BSD)
1985 :	Unix System V
1988 :	Minix
1992 :	Linux

Linux sera le système principalement utilisé en TP d'informatique, du L1 au M2

Unix : architecture



Zoom sur le noyau



Le *Shell* Unix

- Interface utilisateur *en mode texte*

L'utilisateur écrit des commandes dont le résultat est affiché à l'écran

- Interface « historique » sous Unix
- Expose à l'utilisateur certains concepts Unix (permissions, propriétaires, processus, ...)
- Ces concepts sont importants pour pouvoir concevoir des programmes (lire et écrire dans des fichiers, effectuer des connexions réseaux, ...)

Le *Shell* Unix

Exemple de session *shell*:

```
$ ls
Documents   Downloads   Public    Person
$ cd Documents
$ ls
compte_rendu.txt
$ mv compte_rendu.txt cr.txt
$ ls
cr.txt
```

Plan

1 Présentation du cours ✓
2 Le système Unix
2.1 Principes des systèmes d'exploitation ✓
2.2 Système de gestion de fichiers

Système de gestion de fichiers (*filesystem*)

- *Organise* les données sur le support physique
- Protège contre les *corruptions de données*
- Gestion optimale de l'espace disponible
- *Accès efficace* aux données
- *Abstraction* du support physique (Disque optique, mémoire flash, disque réseau, ...)
- Enregistrement des *méta-données* (date de création, propriétaire, taille, ...)

Le concept de *fichier*

Un fichier est une *collection d'informations numériques* réunies sous un même *nom* et enregistrée sur un support de stockage

- Manipulable comme une unité
- Selon les systèmes, le *nom* a plus ou moins d'importance
- possède un type

Le concept de *fichier*

Ne pas confondre:

- type du fichier: il influe sur le comportement du système (fichier « normal », répertoire, lien (raccourcis), fichier système, ...). C'est une méta-donnée conservée par le système de fichier
- type du contenu: le type des *données* contenues dans le fichier:
  - DOS puis Windows: l'extension (les 3 derniers caractères après le « . ») détermine le type de contenu
  - MacOS puis OS X/iOS: les premiers octets du fichier déterminent son type
  - Premiers octets ou extension, selon les interfaces utilisées

Les attributs d'un fichier

Nom :
Propriétaire : utilisateur qui possède ce fichier
Groupe : groupe d'utilisateurs qui possède ce fichier
Emplacement : localisation du fichier sur le support physique
Taille : en octet (peut être la taille réelle ou la taille occupée sur le support)
Permissions : « qui a quel droit » sur le fichier (lecture, écriture, exécution, ...)
Type :
Dates : dernier accès, dernière modification, création, ...

Organisation logique des fichiers

Usuellement, les fichiers sont regroupés en *répertoires*. Les répertoires sont imbriqués les uns dans les autres de manière a former une *arborescence*.

Sous Unix il y a un répertoire racine, « / » (*slash*) qui contient toute l'arborescence du système.
Chaque utilisateur possède aussi un répertoire personnel

Noms de fichiers et chemins

Un chemin est une *liste de répertoire* à traverser pour atteindre un fichier ou répertoire donné. Sous Unix, le séparateur de chemin est le « / »
Les chemins absolus commencent par un `/` et dénotent des fichiers à partir de la racine. Exemple:

```
/home/kim/Documents/IntroInfo/cours01.pdf
```

Les chemins relatifs dénotent des fichiers à partir du répertoire courant. Exemple:

```
Documents/IntroInfo/cours01.pdf
```

si on se trouve dans le répertoire `/home/kim`

Les noms spéciaux:

- `.` : dénote le répertoire courant
- `..` : dénote le répertoire parent
- `~` : dénote le répertoire de l'utilisateur courant
- `~toto` : dénote le répertoire de l'utilisateur `toto`

Utilisation du *Shell*

Le *shell* affiche un *invite de commande* (*prompt*). Exemple:

```
kim@machine $
```

On peut alors saisir une commande:

```
kim@machine $ ls *.txt
```

Le shell affiche la *sortie* de la commande:

```
fichier1.txt   fichier2.txt
```

Certains caractères doivent être précédés d'un « \ » (échappés):

```
kim@machine $ ls mon\ fichier\#1.txt
```

La ligne de commande

Une ligne de commande a la forme :
`prog item1 item2 item3 item4 …`

1. Si `prog` est un chemin il doit dénoter *un fichier exécutable*
2. Si `prog` est un simple nom, il doit dénoter un fichier exécutable se trouvant dans un des *répertoires prédéfinis* (`/bin`, `/usr/bin`, …)
3. Pour chaque $\texttt{item}_i$ (séparés par un ou plusieurs espaces non échappés) le *shell* fait une *expansion de nom*
4. La liste de toutes les chaînes de caractères expansées est passée comme argument au programme `prog`

Expansion des noms / Motifs glob

Certains caractères sont *interprétés* de manière spéciale par le *shell*. Ces caractères sont « expansés » selon des règles. Si la forme *expansée* correspond a un ou plusieurs fichiers existants, alors leurs noms sont placés sur la ligne de commande. Sinon la chaîne de caractère de départ garde sa valeur textuelle.

Motifs glob

Règles d'expansion: `*` n'importe quelle chaîne
`?` n'importe quel caractère
`[ab12…]` un caractère dans la liste
`[^ab12…]` un caractère absent de liste
`[a-z]` un caractère dans l'intervalle
`[^a-z]` un caractère absent de l'intervalle

$?(m_1|\ldots|m_n)$ $@(m_1|\ldots|m_n)$ $*(m_1|\ldots|m_n)$ $+(m_1|\ldots|m_n)$
k motifs parmi les $\texttt{m}_i$:

- `?` : $0 \leq k \leq 1$
- `@` : $k = 1$
- `*` : $k \geq 0$
- `+` : $k \geq 1$

$!(m_1|\ldots|m_n)$: ni $\texttt{m}_1$, ..., ni $\texttt{m}_n$

Motifs glob : exemples

On suppose que le répertoire courant contient les fichiers :

```
ficher1.txt   fichier2.txt   PERSO.txt   une_image.jpeg
```

- `ls *.txt`

```
ficher1.txt   fichier2.txt   PERSO.txt
```

(n'importe quel nom qui termine par `.txt`)

- `ls *[0-9]*`

```
ficher1.txt   fichier2.txt
```

(n'importe quel nom de fichier qui contient un chifffre)

- `ls [a-z]*`

```
ficher1.txt   fichier2.txt   une_image.jpeg
```

(n'importe quel nom de fichier qui commence par une minuscule)

Motifs glob : exemples avancés

On suppose que le répertoire courant contient les fichiers :

```
ficher1.txt   fichier2.txt    PERSO.txt     une_image.jpeg
```

- `ls *([a-z])`

```
ls: cannot access '*([a-z])': No such file or directory
```

(tous les noms contenant uniquement des minuscules. Il n'y en a pas, donc le shell pass literalement le nom de fichier *`*([a-z])`* en argument à la commande `ls` qui affiche un message d'erreur.)

- `ls +([^A-Z]).+([a-z])`

```
ficher1.txt   fichier2.txt   une_image.jpeg
```

(tous les noms contenant un point et tel qu'avant le point il y a une suite de caractères non vide dont aucun n'est une majuscule et après le point il y a une suite non vide de minuscules)

Commandes shell de base

- *cd chemin* : *chemin* devient le répertoire courant. Si absent, utilise le répertoire personnel
- *ls* $chemin_1$ ... $chemin_n$: affiche le nom des *n* fichiers. Si *n=0* affiche le contenu du répertoire courant. Avec l'option *-l* affiche la liste détaillée.
- *cp* $chemin_1$ $chemin_2$: copie de fichier
- *mv* $chemin_1$ $chemin_2$: déplacement de fichier (et renommage)
- *rm* $chemin_1$ ... $chemin_n$: supprime les fichiers (définitif)
- *mkdir nom* : crée le répertoire *nom*
- *mkdir -p chemin* : crée le répertoire dénoté par le *chemin* ainsi que tous les répertoires intermédiaires.

Droits et propriétés des fichiers

Sous Unix un utilisateur est identifié par son *login* (ou nom d'utilisateur). Chaque utilisateur est dans un *groupe principal*.

Au SIF (Service Informatique des Formations, bât. 336) :

- Votre login Unix est de la forme `prenom.nom` (le même que votre adresse email sans le `@universite-paris-saclay.fr`, avec le même mot de passe).
- Il existe aussi un alias plus court (généralement première lettre du prénom, partie du nom de famille puis un chiffre). Cela permet de taper :

```
gdehom7
```

plutôt que

```
guillaume-emmanuel.de-homem-christo
```

avant de commencer son TP (cf. feuille de TP pour récupérer son login court).

Droits et propriétés des fichiers (suite)

Chaque fichier appartient à un utilisateur et à un groupe.
Chaque fichier possède 3 permissions pour son propriétaire, son groupe et tous les autres. Les permissions sont lecture, écriture, exécution (plus d'autres non abordées dans ce cours).

Permission	**fichier**	**répertoire**
lecture (r)	lire le contenu du fichier	lister le contenu du répertoire
écriture (w)	écrire dans le fichier	supprimer/renommer/créer des fichiers dans le répertoire
exécution (x)	exécuter le fichier (si c'est un programme)	rentrer dans le répertoire

```
$ ls -l
drwxr-x---  9 kim prof 4096 Sep  7 21:31 Documents
```

La commande *chmod*

`chmod` *permissions* $chemin_1$ … $chemin_n$

modifie les permissions des fichiers *1* à *n*. La chaîne *permissions* est soit une suite de modifications de permissions *symbolique* soit l'ensemble des permissions données de manière *numérique*:

```
chmod 755 fichier.txt
chmod u-w,a+x,g=w fichier.txt
```


Permissions numériques

On groupe les *bits* de permissions par trois puis on convertit en décimal:

Utilisateur			Groupe			Autres		
r	w	x	r	w	x	r	w	x
1	1	0	1	0	0	0	0	0
6			4			0		

Dans cet exemple 110_2 (lu en base 2) fait 6 en base 10, 100_2 fait 4 et 000_2 fait 0.
Rappel : $110_2 = 1\times2^2 + 1\times2^1 + 0\times2^0 = 6$
Le fichier est lisible et modifiable mais pas exécutable par son propriétaire, lisible pour le groupe. Les autres ne peuvent ni le lire ni le modifier.

Permissions symboliques

```
cible modifieur permission
```

- *cible* : u (utilisateur), g (groupe), o (others), a (all)
- *modifieur* : + (autorise), - (interdit), = (laisse inchangé)
- *permission* : r (lecture), w (écriture), x (exécution)

Exemple:

```
chmod u+rw,u-x,g+r,g-wx,o-rwx fichier.txt
```


Liens symboliques (1)

Pour des raisons d'organisation, on veut pouvoir « voir » le même fichier ou répertoire sous deux noms différents (ou à deux endroits différents). Par exemple:

```
$  ls -l Documents/Cours
total 8
drwxr-xr-x 3 kim prof 4096 Sep  9 11:30 Licence
drwxr-xr-x 3 kim prof 4096 Sep  9 11:30 Master

$  cd Documents/Cours/Master; ls
 Compilation    LangagesDynamiques

$  cd LangagesDynamiques; ls
cours01 cours02  cours03 cours04 cours05  cours06   Prereq

$ ls -l Prereq
lrwxrwxrwx 1 kim prof 28 Sep  9 11:30 Prereq -> ../../Licence/IntroInfo/
```

Liens symboliques (2)

La commande *ln* permet de créer des *liens symboliques*. Un lien est un petit fichier qui contient un *chemin* vers un fichier de destination.

Exemple d'utilisation

```
$ ln -s ../foo/bar/baz/toto.txt rep/titi.txt
```

crée un lien vers le fichier `toto.txt` sous le nom `titi.txt` (chacun placé dans des sous/sur répertoires)

- Ouvrir/modifier le lien ⇒ ouvre/modifie la cible
- Supprimer le lien ⇒ supprime le lien mais pas la cible
- Si la cible est un répertoire, faire `cd` nous place « dans » la cible, mais le répertoire parent est celui d'où l'on vient

Cela permet de créer l'illusion que la cible a été copiée à l'identique, sans les inconvénients
Sous Windows : les liens s'appellent des « raccourcis »

À propos de la suppression

La commande `rm fichier` efface un fichier définitivement
La commande `rm -d rep` efface un répertoire s'il est vide
La commande `rm -r rep` efface un répertoire récursivement mais demande confirmation avant d'effacer des éléments
La commande `rm -rf rep` efface un répertoire récursivement et sans confirmation

Toute suppression est définitive

Gag classique :

```
$  mkdir \~
...
$ ls
  Documents  Photos  Musique   ~
$ rm -rf ~
```

☠ ☠ ☠ ☠ ☠ ☠

Obtenir de l'aide sur une commande

La commande *man* permet d'obtenir de l'aide sur une commande. Lors qu'une page d'aide est affichée, on peut la faire défiler avec les touches du clavier, la quitter avec « `q` » et rechercher un mot avec la touche « `/` »

```
LS(1L)            Manuel de l'utilisateur Linux            LS(1L)
NOM
       ls, dir, vdir - Afficher le contenu d'un répertoire.
SYNOPSIS
       ls [options] [fichier...]
       Options POSIX : [-1acdilqrtuCFR]
       Options  GNU  (forme courte) : [-1abcdfgiklmnopqrstuxABCD
       FGLNQRSUX] [-w cols] [-T cols]  [-I  motif]  [--full-time]
       [--format={long,verbose,commas,across,vertical,single-col
       umn}]                  [--sort={none,time,size,extension}]
       [--time={atime,access,use,ctime,status}]
       [--color[={none,auto,always}]] [--help] [--version] [--]
DESCRIPTION
       La commande ls affiche  tout  d'abord  l'ensemble  de  ses
       arguments  fichiers  autres  que  des répertoires. Puis ls
       affiche  l'ensemble  des  fichiers  contenus  dans  chaque
       répertoire  indiqué.   dir et vdir sont des versions de ls
       affichant par défaut leurs résultats  avec  d'autres  for
       mats.
```


Recherche de fichiers

La commande `find rep` *critères* permet de trouver tous les fichiers se trouvant dans le répertoire `rep` (ou un sous répertoire) et répondant à certains critères. Exemples de critères :

- `-name '*toto*'` dont le nom contient `toto`
- `-iname '*toto*'` pareil, mais insensible à la casse
- `-size +200M` dont la taille sur le disque est supérieure à 200 Mo
- c_1 `-a` c_2 pour lesquels les critères c_1 et c_2 sont vrais
- c_1 `-o` c_2 pour les-quel l'un au moins des critères c_1 et c_2 est vrais
- `-user toto` qui appartiennent à l'utilisateur `toto`
- `-exec` *cmd* `{} \;` pour exécuter *cmd* sur chaque fichié trouvé. La chaîne `{}` est remplacée par le nom de fichier et `\;` sert à marquer la fin de commande.

Comment trouver toutes les options de la commande `find` ? *man find*

Recherche de fichiers (exemples)

Trouver tous les fichiers (dans un sous-répertoire) du répertoire courant dont le *nom se finit par `.jpg`* et dont la taille est supérieure à 1 Mo

```
find . -name '*.jpg' -a -size +1M
```

Trouver tous les fichiers (dans un sous-répertoire) du répertoire courant dont le *nom se finit par `.mpg` (sans tenir compte de la casse)* et dont la taille est supérieure à 10 Mo, et rajouter l'extension .bak à ces fichiers

```
find . -iname '*.mpg' -a -size +10M -exec mv {} {}.bak \;
```


Quelques commandes utiles

- `cat` *`fichier`* : permet d'afficher le contenu d'un fichier dans le terminal
- `less` *`fichier`* : permet de lire le contenu d'un fichier (avec défilement en utilisant les flèches du clavier si le fichier est trop grand)
- `sort` *`fichier`* : permet d'afficher les lignes d'un fichier triées (on peut spécifier des options de tri)
- `file` *`fichier`* : permet de connaître le type d'un fichier
- `wc` *`fichier`* : permet de compter le nombre de caractères/mots/lignes d'un fichier
- `head` *`fichier`* : permet de garder les *n* premières lignes d'un fichier

On verra comment composer ces commandes pour exécuter des opérations complexes

Shell et entrées/sorties

Dans le *shell*, l'opérateur | permet d'enchaîner la sortie d'un programme avec l'entrée d'un autre:

```
$ ls -l *.txt | sort -n -r -k 5 | head -n 1
```

1. affiche la liste détaillée des fichiers textes
2. trie (et affiche) l'entrée standard par ordre numérique décroissant selon le 5ème champ
3. affiche la première ligne de l'entrée standard

```
-rw-rw-r    1 kim kim 1048576 Sep 24 09:20 large.txt
```


Fonctionnement des redirections

`cmd < fichier`: *fichier* est ouvert en lecture avant le lancement de *cmd*, le contenu est redirigé vers l'entrée standard de *cmd*.
`cmd > fichier`: *fichier* est ouvert en écriture avant le lancement de *cmd*. Si *fichier* n'existe pas il est créé. S'il existe il est tronqué à la taille 0. La sortie standard de *cmd* est redirigée vers *fichier*.
`cmd >> fichier`: *fichier* est ouvert en écriture avant le lancement de *cmd*. Si *fichier* n'existe pas il est créé. S'il existe, le curseur d'écriture est placé en fin de fichier. La sortie standard de *cmd* est redirigée vers *fichier*.
`cmd 2> fichier`: Comme `>` mais avec la sortie d'erreur
`cmd 2>> fichier`: Comme `>>` mais avec la sortie d'erreur

Attention à l'ordre d'exécution !

Quelques exemples de commandes problématiques :

```
$ sort fichier.txt > fichier.txt
```

`fichier.txt` devient vide ! Il est ouvert en écriture et tronqué avant l'exécution de la commande.

```
$ sort < fichier.txt > fichier.txt
```

`fichier.txt` devient vide ! Il est ouvert en écriture et tronqué avant l'exécution de la commande.

```
$ sort < fichier.txt >> fichier.txt
```

`fichier.txt` contient son contenu original, suivi de son contenu trié !

```
$ cat < fichier.txt >> fichier.txt
```

`fichier.txt` est rempli jusqu'à saturation de l'espace disque !

Quelques explications (1/2)

La commande `sort` doit trier son entrée standard. Elle doit donc la lire intégralement avant de produire la moindre sortie. Pour

```
$ sort < fichier.txt >> fichier.txt
```

on a donc :

1. Ouverture de *fichier.txt* en lecture
2. Ouverture de *fichier.txt* en écriture, avec le curseur positionné en fin
3. Lecture de toute l'entrée
4. Écriture de toute la sortie en fin de *fichier.txt*

Quelques explications (2/2)

La commande `cat` ré-affiche son entrée standard sur sa sortie standard. Elle peut donc lire le fichier morceaux par morceaux et les afficher au fur et à mesure. Supposons que *fichier.txt* contient `AB` :

```
$ cat < fichier.txt >> fichier.txt
```

1. Ouverture de *fichier.txt* en lecture
2. Ouverture de *fichier.txt* en écriture, avec le curseur positionné en fin
3. Lecture de A (et positionnement du curseur de lecture sur B)
4. Écriture de A en fin de fichier *fichier.txt*
5. Lecture de B (et positionnement du curseur de lecture sur A)
6. Écriture de B en fin de fichier *fichier.txt*
7. Lecture de A (et positionnement du curseur de lecture sur B)
8. Écriture de A en fin de fichier *fichier.txt*
9. ...

Conseils...

On évitera toujours de manipuler le même fichier en entrée et en sortie. Il vaut mieux rediriger vers un fichier temporaire, puis renommer ce dernier (avec la commande `mv`).