

# INF203 - Cours 1

## Unix, système de fichiers et commandes de base

Responsable : [romain.couillet@univ.grenoble-alpes.fr](mailto:romain.couillet@univ.grenoble-alpes.fr)

Amphis : Romain Couillet et Benjamin Wack

TDs / TPs : Tom Besson, Florent Bouchez-Tichadou, Basile  
Dubois-Bonnaire, Jean-Loup Haberbush, Thibaut Lopez, Kahina  
Ouazine, Thomas Rahab Lacroix, Claudia Rusu

Université Grenoble Alpes

# Plan

- 1 Présentation de l'UE
- 2 Interpréteur
- 3 Système de fichiers
- 4 Commandes
- 5 Métacaractères (Wild cards)
- 6 Gestion des tâches
- 7 Droits d'accès

# Plan

- 1 **Présentation de l'UE**
- 2 Interpréteur
- 3 Système de fichiers
- 4 Commandes
- 5 Métacaractères (Wild cards)
- 6 Gestion des tâches
- 7 Droits d'accès

# Programme

- ❶ **Système UNIX** : interpréteur de commandes (utilisation et programmation), système de fichiers
- ❷ **Programmation en C** : structure du langage, interface de programmation fournie par le système
- ❸ **Outils de développement** : clang
- ❹ **Automates** : découverte du formalisme, application à l'interprétation
- ❺ **Programmation d'un mini-interpréteur** : en utilisant toutes les notions précédentes



*Ken Thompson, l'inventeur d'UNIX et C.*

# Pourquoi apprendre UNIX, C en 2021 ?

- **Pour comprendre le cœur d'un OS** : I/O et programmation bas niveau
- **Pour des raisons historiques** : comprendre l'évolution des architectures
- **Bonnes pratiques** : prise de recul sur le lien entre mémoire, processeur et I/O
- **Universalité et optimalité** : nombreux programmes écrits en C pour des raisons d'optimisation (proximité machine)

# Pourquoi apprendre UNIX, C en 2021 ?

- **Pour comprendre le cœur d'un OS** : I/O et programmation bas niveau
- **Pour des raisons historiques** : comprendre l'évolution des architectures
- **Bonnes pratiques** : prise de recul sur le lien entre mémoire, processeur et I/O
- **Universalité et optimalité** : nombreux programmes écrits en C pour des raisons d'optimisation (proximité machine)
- **Parce que ce sont des outils "conviviaux"** :
  - on comprend comment fonctionne la machine, les programmes
  - on peut réparer, modifier, créer par nous-mêmes (logiciel libre)
  - offre beaucoup de **résilience** et un **sentiment de contrôle**.



*Ivan Illich et la convivialité.*

# Organisation

**6h par semaine** : 1h30 Cours, 1h30 TD et 3h TP

- Seulement 2h de TP encadrées (le reste en autonomie)
- TPs en **binômes**
- **Compte-rendu de TP** (1 par binôme) à rendre avant la séance de TD suivante sur Caséine  
→ **Inscrivez-vous au plus vite sur Caséine (INF203).**
- Les comptes-rendus ne sont **pas notés** mais sont régulièrement contrôlés.
- En supplément, activités pratiques sur Caséine pour s'exercer (30min/semaine).

# Évaluation

- **CC1** : un DS durant la semaine de partiels (40 %)
- **CC2** : projet `cowsay` : vous pouvez débiter dès maintenant ! (10 %)
- **Examen terminal** (50 %, deux sessions)

+ règle du MAX : note à l'UE = note à l'examen si elle est meilleure



# Documentation

Documents en ligne à l'adresse

<https://inf203.gricad-pages.univ-grenoble-alpes.fr>

- documents généraux et bibliographie
- slides de cours
- feuilles d'exercices
- sujets de TP
- sujet du projet `cowsay`
- fichiers pour les TPs
- annales d'examen

**À consulter régulièrement car fréquemment mis à jour**

# Plan

- 1 Présentation de l'UE
- 2 Interpréteur**
- 3 Système de fichiers
- 4 Commandes
- 5 Métacaractères (Wild cards)
- 6 Gestion des tâches
- 7 Droits d'accès

# Système d'exploitation

Noyau et ensemble de programmes permettant l'**interaction** entre un **utilisateur** et la **machine**, il gère en particulier les ressources

- de stockage (disques, clés, cartes mémoire)
- de calcul (processeurs/coeurs, accélérateurs)
- mémoire (mémoire vive, swap, cache de disque)
- réseau
- d'affichage et d'impression
- ...

**Exemple** : Windows, Unix (Linux, BSD, OSX), Android, ...

# Interpréteur

Un système d'exploitation offre à l'utilisateur des moyens d'**interagir** avec lui, comme par exemple un **interpréteur de commandes** textuel

- attend et exécute des **commandes** saisies par l'utilisateur
- fourni un **retour** à l'utilisateur (affichage, messages d'erreur)

Alternativement, l'interface peut être **graphique** (fenêtres, icônes, ...), dans ce cas les interactions sont (paradoxalement) limitées

- **pas d'automatisation** des tâches
- **ensembles de ressources limités** à une zone sélectionnable


# Shell

**Interpréteur de commandes textuel** fourni de manière standard par tous les UNIX

- **sh, Bourne shell, interpréteur historique des premiers UNIX**
- `csh`, `tcsh`, famille non compatible avec Bourne, peu utilisés
- `ksh`, `zsh`, `ash`, `dash`, . . . , plus ou moins vieille famille compatible avec Bourne
- **`bash`, compatible avec Bourne, par défaut sur la majorité des UNIX actuels**

Pour cette UE, nous apprendrons `sh` (portable, pas trop de notions) et utiliserons `bash`.

# Fonctionnement de l'interpréteur

- ❶ Affichage de l'**invite** de commande (prompt) :  
im2ag-turing:~\$
- ❷ Attente d'une séquence de caractères terminée par   
im2ag-turing:~\$ echo Hello World ! 
- ❸ Analyse de cette séquence : est-ce une commande correcte ?
  - si oui, on l'exécute, et on attend que cette exécution se termine !
  - sinon, on affiche un message d'erreur

Ici, la commande est correcte donc exécution :

affichage de «Hello World !»

# Fonctionnement de l'interpréteur

- ➊ Affichage de l'**invite** de commande (prompt) :  
im2ag-turing:~\$
- ➋ Attente d'une séquence de caractères terminée par   
im2ag-turing:~\$ echo Hello World ! 
- ➌ Analyse de cette séquence : est-ce une commande correcte ?
  - si oui, on l'exécute, et on attend que cette exécution se termine !
  - sinon, on affiche un message d'erreur

Ici, la commande est correcte donc exécution :

affichage de «Hello World !»

**Remarque** : exécuter une commande signifie (en général) démarrer un programme chargé d'exécuter cette commande

# Messages d'erreur

Les commandes Unix sont toujours de la même forme :

`<nom de commande>` suivi de 0 ou plusieurs `<arguments>`

## Exemples :

```
cp tp1.c tp2.c
mkdir TP
ls
```

Lorsqu'une commande est incorrecte, le shell ou la commande fournit un message d'erreur, différent selon la nature de l'erreur

- `<nom de commande>` incorrect (message du shell)  
`-bash: salut : commande introuvable`
- `<arguments>` incorrects (message de la commande)  
`cp: missing file operand`
- la commande ne peut s'exécuter correctement  
`touch: cannot touch '/INF203': Permission denied`

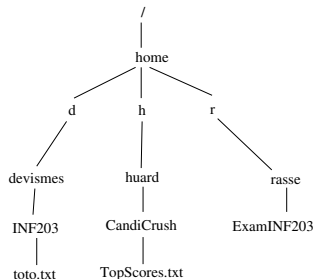


# Plan

- 1 Présentation de l'UE
- 2 Interpréteur
- 3 Système de fichiers**
- 4 Commandes
- 5 Métacaractères (Wild cards)
- 6 Gestion des tâches
- 7 Droits d'accès

# Arborescence du système de fichiers

Tout objet de stockage (fichier ou répertoire) est identifié par sa place dans l'arborescence.



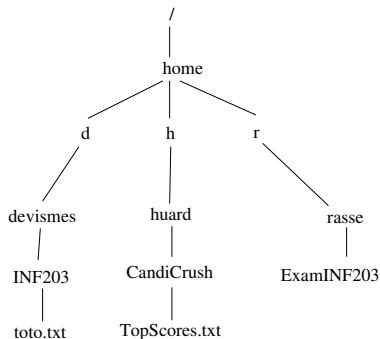
Relation père/fils

- /      répertoire *racine*
- .      répertoire courant
- ..     répertoire parent
- ~      répertoire principal (de login)  
         de l'utilisateur courant
- ~toto    répertoire principal de *toto*

# Chemin absolu

Chemin dans l'arbre **depuis la racine** (/)  
jusqu'à l'objet recherché (fichier ou un  
répertoire) exprimé par une liste de  
répertoires (séparé par des /)

Aller-retour possible !



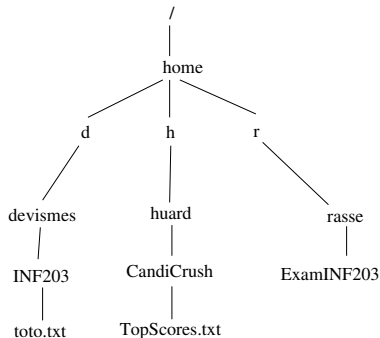
# Chemin absolu

Chemin dans l'arbre **depuis la racine** (/)  
jusqu'à l'objet recherché (fichier ou un  
répertoire) exprimé par une liste de  
répertoires (séparé par des /)

Aller-retour possible !

## Exemples :

- fichier toto.txt :  
/home/d/devismes/INF203/toto.txt
- répertoire CandiCrush :  
/home/h/huard/CandiCrush  
/home/h/huard/.. /huard /CandiCrush



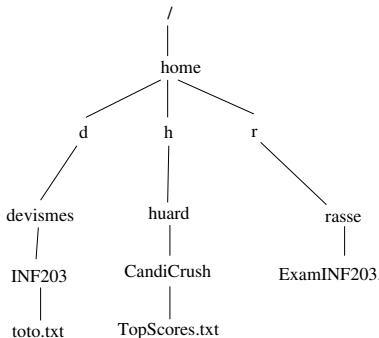
# Chemin absolu

Chemin dans l'arbre **depuis la racine** (/)  
jusqu'à l'objet recherché (fichier ou un  
répertoire) exprimé par une liste de  
répertoires (séparé par des /)

Aller-retour possible !

## Exemples :

- fichier toto.txt :  
/home/d/devismes/INF203/toto.txt
- répertoire CandiCrush :  
/home/h/huard/CandiCrush  
/home/h/huard/.. /huard /CandiCrush

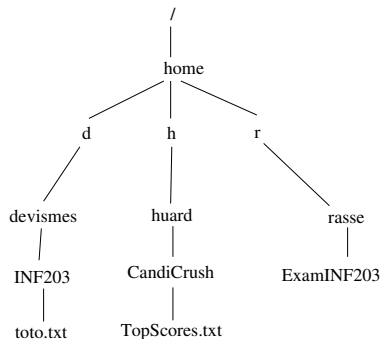


**Remarque :** Il existe une infinité de chemins, en général on choisit le plus court

# Chemin relatif

Chemin dans l'arbre **depuis le répertoire courant** jusqu'à l'objet recherché (fichier ou un répertoire) exprimé par une liste de répertoires (séparé par des /)

Aller-retour possible !



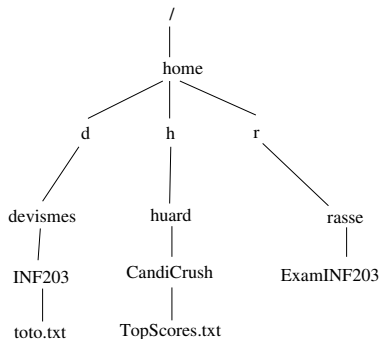
# Chemin relatif

Chemin dans l'arbre **depuis le répertoire courant** jusqu'à l'objet recherché (fichier ou un répertoire) exprimé par une liste de répertoires (séparé par des /)

Aller-retour possible !

**Exemples** : Si répertoire courant est *devismes*, le fichier *ExamINF203.pdf* peut être désigné par

- `../../r/rasse/ExamINF203.pdf`
- `../../h/huard/CandiCrush/../../r/rasse/ExamINF203.pdf`



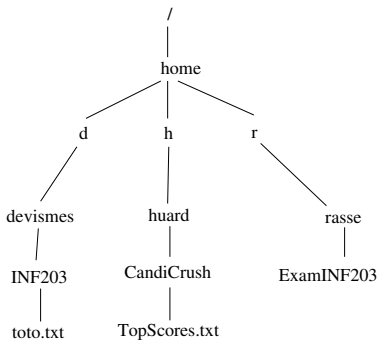
# Chemin relatif

Chemin dans l'arbre **depuis le répertoire courant** jusqu'à l'objet recherché (fichier ou un répertoire) exprimé par une liste de répertoires (séparé par des /)

Aller-retour possible !

**Exemples :** Si répertoire courant est *devismes*, le fichier *ExamINF203.pdf* peut être désigné par

- `../../r/rasse/ExamINF203.pdf`
- `../../h/huard/CandiCrush/../../r/rasse/ExamINF203.pdf`



**Remarque :** En pratique, on utilise souvent un chemin relatif quand on est proche, sinon on utilise le chemin absolu.



# Nom de fichier

- **Unix différencie les majuscules des minuscules !**
- **Unix se fiche des extensions**, elles ne déterminent pas le type du fichier. Il s'agit juste d'une « bonne pratique » pour classer les fichiers.
- Éviter les espaces, caractères spéciaux et accents dans les noms de fichiers :
  - difficile à gérer dans les scripts SHELL,
  - pas portable, *e.g.* encodage (isolatin, UTF8 ...) des caractères accentués

# Plan

- 1 Présentation de l'UE
- 2 Interpréteur
- 3 Système de fichiers
- 4 Commandes**
- 5 Métacaractères (Wild cards)
- 6 Gestion des tâches
- 7 Droits d'accès

# Syntaxe générale

```
commande [-options ] [arguments ] arguments
```

La commande peut être suivie d'**options** ou d'**arguments** séparés par des espaces.

Si ceux-ci apparaissent entre crochets dans l'aide en ligne **man**, c'est qu'ils sont facultatifs, sinon ils sont obligatoires.

Les options sont précédées d'un « - » contrairement aux arguments

# Syntaxe générale

commande [-options ] [arguments ] arguments

La commande peut être suivie d'**options** ou d'**arguments** séparés par des espaces.

Si ceux-ci apparaissent entre crochets dans l'aide en ligne **man**, c'est qu'ils sont facultatifs, sinon ils sont obligatoires.

Les options sont précédées d'un « - » contrairement aux arguments

**Exemple** : lister des fichiers `ls`

`ls [-altrR] [noms... ]`

-a : (all) tous les fichiers, même cachés

-l : (long) lister au format long

-t : (tri) lister en triant par date

-R : (recursive) lister récursivement dans les répertoires

`ls -l ; ls -ltr ; ls -R -l /usr ; ls -al . ; ls -lR /etc`

# Aide en ligne

L'aide en ligne **man** contient entre autres la description :

- des commandes UNIX (section 1)
- des fonctions C (section 3)

**Exemple** : description de la commande `ls`

```
man ls
```

Quelques séquences de touches utiles

- `/<motif>` pour la recherche (puis `p` et `n`) pour précédent et suivant
- `Ctrl-b` et `Ctrl-f` pour reculer et avancer
- `q` pour quitter

Une commande UNIX et une fonction C peuvent avoir le même nom, *e.g.* `printf`. On précise alors la section :

- commande UNIX : `man 1 printf`
- fonction C : `man 3 printf`

# Sur les répertoires

`pwd` : affiche le répertoire courant (print working directory)

# Sur les répertoires

`pwd` : affiche le répertoire courant (`print working directory`)

`cd` : change de répertoire courant (`change directory`)

- `cd . .` remonte au répertoire père
- `cd /` va dans le répertoire racine
- `cd` va dans le répertoire principal (*home*) de l'utilisateur courant
- `cd -` va dans le répertoire exploré précédemment

# Sur les répertoires

`pwd` : affiche le répertoire courant (`print working directory`)

`cd` : change de répertoire courant (`change directory`)

- `cd ..` remonte au répertoire père
- `cd /` va dans le répertoire racine
- `cd` va dans le répertoire principal (*home*) de l'utilisateur courant
- `cd -` va dans le répertoire exploré précédemment

`mkdir` : `mkdir rep` crée le répertoire `rep` (`make directory`)



# Sur les répertoires

`pwd` : affiche le répertoire courant (`print working directory`)

`cd` : change de répertoire courant (`change directory`)

- `cd ..` remonte au répertoire père
- `cd /` va dans le répertoire racine
- `cd` va dans le répertoire principal (*home*) de l'utilisateur courant
- `cd -` va dans le répertoire exploré précédemment

`mkdir` : `mkdir rep` crée le répertoire `rep` (`make directory`)

`rmdir` : `rmdir rep` supprime le répertoire `rep` **s'il est vide** (`remove directory`)

# Sur les fichiers

`ls` : liste les fichiers et répertoires (list)

# Sur les fichiers

`ls` : liste les fichiers et répertoires (`list`)

`cp` : copie des fichiers et répertoires (`copy`)

# Sur les fichiers

`ls` : liste les fichiers et répertoires (`list`)

`cp` : copie des fichiers et répertoires (`copy`)

`rm` : supprime des fichiers et répertoires (`remove`)

# Sur les fichiers

`ls` : liste les fichiers et répertoires (`list`)

`cp` : copie des fichiers et répertoires (`copy`)

`rm` : supprime des fichiers et répertoires (`remove`)

`mv` : déplace et/ou renomme un fichier ou un répertoire (`move`)

# Sur les fichiers

`ls` : liste les fichiers et répertoires (`list`)

`cp` : copie des fichiers et répertoires (`copy`)

`rm` : supprime des fichiers et répertoires (`remove`)

`mv` : déplace et/ou renomme un fichier ou un répertoire (`move`)

`touch` : crée un fichier vide (ou change la date de dernier accès d'un fichier existant)

# Contenu de fichiers

`cat` (**concaténer**) : permet de concaténer et d'afficher des fichiers.


```
cat fich  
cat fich1 fich2
```

# Contenu de fichiers

`cat` (**concaténer**) : permet de concaténer et d'afficher des fichiers.

```
cat fich
```

```
cat fich1 fich2
```

`less` (**ou more**) : affiche des fichiers **page par page** ()


```
less fich
```



# Contenu de fichiers

`cat` (**concaténer**) : permet de concaténer et d'afficher des fichiers.

```
cat fich
cat fich1 fich2
```

`less` (**ou more**) : affiche des fichiers **page par page** ()

```
less fich
```


`file` : permet d'identifier le type d'un fichier.

```
file fich
```

# Contenu de fichiers

`cat` (**concaténer**) : permet de concaténer et d'afficher des fichiers.

```
cat fich
cat fich1 fich2
```

`less` (**ou more**) : affiche des fichiers **page par page** ()

```
less fich
```

`file` : permet d'identifier le type d'un fichier.

```
file fich
```

`vim` (**console**), `emacs`, `gedit` ... : éditeurs de texte

# Copie de fichiers/répertoires

`cp (copy)` : permet

- de copier des fichiers
- des répertoires avec l'option `-R`

**Syntaxe :** `cp source(s) destination`

*source(s)* = ce qui est copié      *destination* = vers où c'est copié

**Exemple :** `cp index.html /home/toto/`

Copie le fichier `index.html` dans le répertoire `toto` (si `toto` existe)

**Attention :**

- *source(s)* = fichiers à copier
- *source(s)* peut être des fichiers/répertoires
- *destination* = où on les copie
- *destination* peut exister ou non, être un répertoire ou un fichier

# Copie de fichiers/répertoires (exemple)

- ❶ `cp fich1 fich2`
- ❷ `cp fich rep`
- ❸ `cp fich .`
- ❹ `cp fich1 ... fichn rep`
- ❺ `cp -R rep1 rep2 (rep2 existe ou non, copie dans rep2 ou nommé rep2)`

# Renommage/déplacement de fichiers

`mv` (**move**) : permet

- de changer le nom des fichiers ou répertoires
- de déplacer des fichiers ou répertoires  
(équivalent à une copie, suivie d'une suppression).

**Syntaxe :** `mv source(s) destination`

## Exemples :

- `mv index.html accueil.html` (renommage)
- `mv index.html /home/site/` (déplacement)
- `mv index.html /home/site/accueil.html`  
(déplacement + renommage)

# Effacement ou suppression de fichiers ou répertoires

`rm (remove)` : permet de supprimer des fichiers ou répertoires  
Option `-R` pour supprimer un répertoire (et son contenu)  
**Attention** : pas de corbeille !

**Syntaxe** : `rm fich`

## Exemples :

- `rm /home/site/index.html`
- `rm -R /home/site`

# Plan

- 1 Présentation de l'UE
- 2 Interpréteur
- 3 Système de fichiers
- 4 Commandes
- 5 Métacaractères (Wild cards)**
- 6 Gestion des tâches
- 7 Droits d'accès

# Expansions

Dans une ligne de commande, le système de fichiers remplace certains caractères par des noms de fichiers  
(au sens large, c'est-à-dire fichiers ordinaires, répertoires ou liens).

Les expansions utilisent les symboles suivants : \*, ?, [ ]



# Métacaractères (1/4)

## Métacaractère \*

Remplacé par n'importe quelle suite de caractères (vide compris) en cohérence avec le système de fichier (non remplacé si aucune correspondance).

### Exemple :

- lister tous les fichiers dont la 1<sup>ère</sup> lettre est un a

```
ls a*
```

- déplacer tous les fichiers `c` du répertoire courant vers le répertoire `source`

```
mv *.c source
```

# Métacaractères (2/4)

## Métacaractère ?

Remplacé par 1 et 1 seul caractère en cohérence avec le système de fichier (non remplacé si aucune correspondance).

**Exemple :** lister tous les fichiers nommés `tp` suivi d'un et un seul caractère (`tp1`, `tp2`, `tp3`, ... mais pas `tp`, `tp12`, `tp1.txt` ...)

```
ls tp?
```

# Métacaractères (3/4)

## Métacaractère [ ]

Remplacé par 1 et 1 seul caractère en cohérence avec le système de fichier dans l'ensemble de caractères donné entre les crochets, séquence de

- caractères de l'ensemble
- intervalles (bornes séparées par —)

L'ensemble est un complément s'il débute par ^

### Exemples :

- Lister tous les fichiers dont la première lettre est a, b ou c

```
ls [abc]* ou ls [a-c]*
```

- Lister tous les fichiers qui se terminent par 7, 8 ou 9

```
ls *[7-9]
```

- Lister tous les fichiers qui ne se terminent pas par 6, 7, 8, 9

```
ls *[!6-9] ou ls *[^6-9]
```

# Métacaractères (4/4)

Lister la liste de fichiers suivante : toto31.txt, toto32.txt, ..., toto39.txt

Quelle est la bonne commande ?

- `ls toto[31-39].txt`
- `ls toto3[1-9].txt`
- `ls [toto31-toto39].txt`

## Métacaractères (4/4)

Lister la liste de fichiers suivante : toto31.txt, toto32.txt, ..., toto39.txt

Quelle est la bonne commande ?

- `ls toto[31-39].txt`
- `ls toto3[1-9].txt`
- `ls [toto31-toto39].txt`

lister la liste de fichiers toto00.txt, toto01.txt, ... toto39.txt

Quelle est la bonne commande ?

- `ls toto??.txt`
- `ls toto[0-3][0-9].txt`

# Exercice

Analysez le comportement de l'interpréteur quand on exécute `cp *` dans les cas suivants :

- le répertoire courant contient 1 fichier (et c'est tout)
- le répertoire courant contient 2 fichiers (et c'est tout)
- le répertoire courant contient 3 fichiers (et c'est tout)
- le répertoire courant contient n fichiers et 1 répertoire (et c'est tout)

# Plan

- 1 Présentation de l'UE
- 2 Interpréteur
- 3 Système de fichiers
- 4 Commandes
- 5 Métacaractères (Wild cards)
- 6 Gestion des tâches**
- 7 Droits d'accès

# Tâche de fond

On peut ne pas vouloir attendre que la commande en cours soit terminée avant de lancer une nouvelle commande :

- parce que c'est une commande qui prend du temps (un calcul long)
- parce que l'on veut que cette commande s'exécute en continu (e.g., navigateur, éditeur de texte)



# Tâche de fond

On peut ne pas vouloir attendre que la commande en cours soit terminée avant de lancer une nouvelle commande :

- parce que c'est une commande qui prend du temps (un calcul long)
- parce que l'on veut que cette commande s'exécute en continu (e.g., navigateur, éditeur de texte)

On peut indiquer à l'interpréteur qu'une commande doit être exécutée en « tâche de fond » (*background*) en la faisant suivre du symbole '&'.

Dans ce cas l'interpréteur de commande lance l'exécution (si la commande est correcte) et se remet immédiatement en attente de la prochaine commande.

**Exemple :** `gedit &`

# Commandes spéciales

- `Ctrl-C (^C)` : arrête l'exécution en cours (premier plan)
- `Ctrl-Z` : suspend l'exécution en cours (premier plan) et redonne la main à la console
- `Ctrl-D` : termine la saisie (ferme le terminal)
- commande `&` : exécute la commande en background (arrière-plan)
- `jobs` : liste des commandes en arrière plan
- `bg` ou `bg %<num>` : reprend et bascule le *job* donné (plus récent par défaut) en arrière plan (si on a oublié `&`, faire `Ctrl-Z` puis `bg`)
- `fg` ou `fg %<num>` : reprend en premier plan l'exécution suspendue ou en arrière plan donnée (plus récent par défaut)
- `kill %<num>` : termine (poliment) l'exécution du *job* donné
- `Ctrl-L` : rafraichit l'écran (*clear*, *reset*)

# Système à temps partagé

## Remarque :

On peut avoir plusieurs commandes qui s'exécutent en même temps  
Il faut que la machine soit capable d'exécuter plusieurs programmes « en même temps » alors qu'elle ne possède en général qu'un (ou un petit nombre de) processeur(s).

# Système à temps partagé

## Remarque :

On peut avoir plusieurs commandes qui s'exécutent en même temps  
Il faut que la machine soit capable d'exécuter plusieurs programmes « en même temps » alors qu'elle ne possède en général qu'un (ou un petit nombre de) processeur(s).

## Remarque :

En pratique cette « simultanéité » est obtenue à travers une partie du système d'exploitation (l'**ordonnanceur** [*scheduler*]) qui permet d'interrompre / relancer l'exécution d'un programme, de gérer une liste de programmes « en attente », de choisir lequel exécuter à un instant donné, *etc.*

# Système à temps partagé

## Remarque :

On peut avoir plusieurs commandes qui s'exécutent en même temps  
Il faut que la machine soit capable d'exécuter plusieurs programmes « en même temps » alors qu'elle ne possède en général qu'un (ou un petit nombre de) processeur(s).

## Remarque :

En pratique cette « simultanéité » est obtenue à travers une partie du système d'exploitation (l'**ordonnanceur** [*scheduler*]) qui permet d'interrompre / relancer l'exécution d'un programme, de gérer une liste de programmes « en attente », de choisir lequel exécuter à un instant donné, etc.

Unix est un système à temps partagé  
(et multi-tâches, multi-utilisateurs)

# Plan

- 1 Présentation de l'UE
- 2 Interpréteur
- 3 Système de fichiers
- 4 Commandes
- 5 Métacaractères (Wild cards)
- 6 Gestion des tâches
- 7 **Droits d'accès**

# Multi-utilisateurs

Unix est un système **multi-utilisateurs** : plusieurs utilisateurs peuvent utiliser le système simultanément, il partagent toutes les ressources (processeur, disque, mémoire, ...).

Cependant les permissions des utilisateurs doivent être restreintes afin de garantir l'intégrité de leurs données et du système lui-même.

Pour le système de fichiers cela se traduit par la notion de **droits d'accès**.

# Catégories

Unix distingue 3 catégories d'utilisateurs :

**User (u)** Désigne la personne qui a créé le fichier/répertoire, c'est le propriétaire (peut être modifié par la suite).

**Group (g)** Désigne les membres du groupe d'utilisateurs.

**Exemples :**

- prof, étudiants, ... pour un établissement d'enseignement
- direction, comptabilité, infographie, ... pour une société.

**Others (o)** Désigne tous les autres utilisateurs.



# Permissions

Unix distingue 3 types de permissions

**r** : read → permission de lecture

**w** : write → permission d'écrire

**x** : execution → permission d'exécuter

Pour visualiser les permissions sur les fichiers/répertoires : `ls -l`

type	u	g	o	#lnk	owner	grp	size	last update	name
d	rwx	r-x	r-x	10	toto	prof	4096	Dec 7 2017	cours
-	rwx	rwx	rwx	1	toto	prof	6770	Nov 8 2012	img.jpg
-	rwx	---	---	1	toto	prof	2001	Nov 1 2012	tp.txt

# Types de fichiers

Il existe de nombreux types de fichiers, les principaux sont :

- : fichier ordinaire

- d : répertoire

- l : lien symbolique (raccourci)

# Sémantique des permissions (fichier)

Fichier	r	Autorise la consultation - affichage - copie
	w	Autorise la modification - modification du contenu - <b>ne permet pas la suppression du fichier !</b> - <b>ne permet pas le changement de nom !</b>
	x	Autorise l'exécution - le fichier doit être un programme

# Sémantique des permissions (répertoire)

Répertoire	r	Autorise la consultation - affichage du contenu ( <code>ls</code> )
	w	Autorise la modification - ajout de contenu ( <i>e.g.</i> , les fichiers et les sous-répertoires) ( <code>touch</code> , <code>mkdir</code> , ...) - suppression de contenu ( <code>rm</code> et <code>rmdir</code> ) - renommage de contenu ( <code>mv</code> )
	x	Autorise la traversée - utilisation dans un chemin d'accès à une entrée (fichier/répertoire) ( <code>cd</code> )

# Modification des droits (1/2)

La commande `chmod` (change mode) permet de changer les droits.

## Syntaxe :

```
      chmod mode cible
      cible : fichier ou répertoire
      mode : qui(ugo) quoi(+ - =) comment(rwx)
              a = all = ugo
      (chmod +r = chmod a+r = chmod ugo+r)
```

## Exemples :

- retirer les droits d'écriture et d'exécution pour le groupe et les autres sur le fichier `img.jpg`

```
      chmod go-wx img.jpg
```

- ajouter les droits de lecture et d'écriture pour le groupe sur le fichier `tp_unix.txt`

```
      chmod g+rw tp_unix.txt
```

# Modification des droits (2/2)

## Syntaxe en forme condensée

`chmod qui&quoi (3 chiffres) fichier/rep`

- 1er chiffre : user
- 2ème chiffre : group
- 3ème chiffre : other

- 1 : exécution
- 2 : écriture
- 4 : lecture

dont la somme donne le droit (puissances de 2  $\Rightarrow$  somme unique).

**Exemple :** `chmod 764 toto.sh`

# Merci de votre attention

