– R108 (I_Linux) –
Les bases du Système d'Exploitation

« Vraiment, je ne suis pas là pour détruire Microsoft. Ça sera juste un effet secondaire tout à fait involontaire. »

Linus Torvalds

1- Introduction : l'informatique et LINUX

Qu'est-ce que l'informatique ?

L'informatique en RT

Le système LINUX : introduction

Qu'est ce l'informatique ?
L'informatique en RT
Le système LINUX : introduction

- Informatique :
  - Art de maîtriser un des outils les plus sophistiqués et puissants créés par l'homme
- Différents niveaux
  - Architecture (matériel), réseaux (communications entre ordinateurs)
  - Systèmes d'exploitation (« langue natale » des ordinateurs)
  - Administration (des ordinateurs, des réseaux)
  - Programmation (stockage, interrogation et traitement des données)

Qu'est ce l'informatique ?
L'informatique en RT
Le système LINUX : introduction

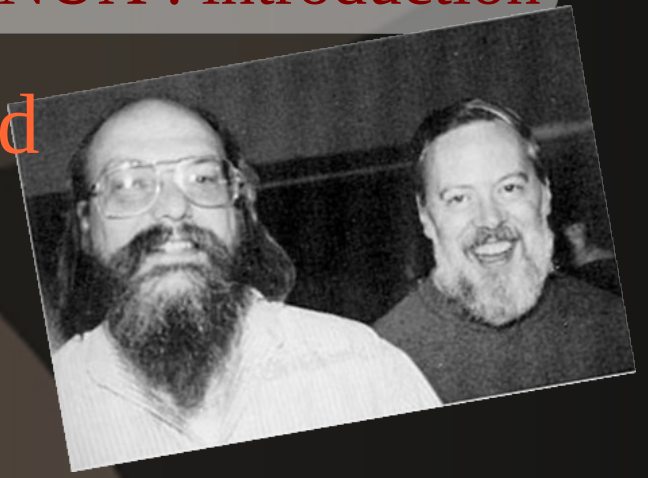
- Quelques conseils pour réussir

  L'informatique c'est parler la langue des ordinateurs

  - Très peu de choses à apprendre
  - Beaucoup de pratique à acquérir

Il faut simplement travailler un peu
… mais très régulièrement

Qu'est ce l'informatique ?
L'informatique en RT
Le système LINUX : introduction



- 30 ans pour passer de Unix à Android
  - 1969 : Naissance de Unix
    (Kenneth Thompson, Dennis Ritchie)
  - 1975 : V7 (commercialisation)
  - Années 80 :
    - 2 types d'Unix : système V (Bell) / BSD (Univ. Berkeley)
    - Différents Unix propriétaires et payants (HP, Sun, …)
  - Années 90 :
    - Normalisation (norme POSIX) et amélioration de l'interface graphique
    - Ouverture du système : LINUX (1991)
  - Années 2000 : base pour Mac OS X, iOS, Android

Qu'est ce l'informatique ?
L'informatique en RT
Le système LINUX : introduction

- Et Linux ?
  - Sur-ensemble de Unix
  - Créé par Linus Torvalds en 1991
  - Initialement pour PC, maintenant partout !
  - Wikipédia : « ce système est né de la rencontre du mode opératoire hacker avec les principes du mouvement du logiciel libre »
  - Nombreux logiciels de développement gratuits tournent sous Linux
  - Plusieurs distributions : Redhat, Debian, …

Qu'est ce l'informatique ?
L'informatique en RT
Le système LINUX : introduction

- Pourquoi apprendre Linux ?
  - Système répandu :
    - Majoritaire sur les super-ordinateurs et les smartphones
    - Très utilisé comme système embarqué (GPS, TV, box, …)
    - Partage le marché avec Windows pour les serveurs informatiques
    - Peu utilisé par « monsieur tout le monde »
  - Architecture mixte souvent utilisée en entreprise :
    - Serveurs sous Linux (web, mail, parefeu, …)
    - Postes de travail sous Windows

Qu'est ce l'informatique ?
L'informatique en RT
Le système LINUX : introduction

- A la fin du module, vous :
  - Aurez assimilé les notions de base d'un système d'exploitation
  - Serez devenu un utilisateur averti de Linux :
    - Commandes de base
    - Expressions régulières
    - Scripts shell
  - Aurez acquis les bases nécessaires pour devenir administrateur système et technicien réseau (cf. TP de réseaux)

2- Les bases de LINUX

L'arborescence

Manipuler les fichiers et les répertoires

Gestion des droits

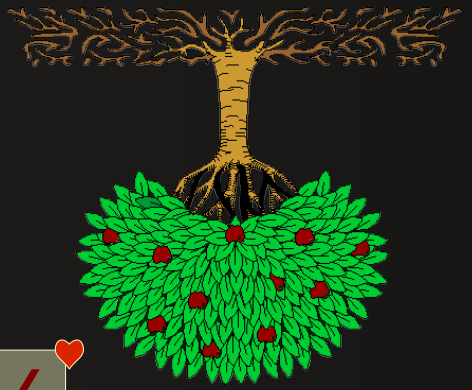
C. Escazut

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

- Fichiers :
  - Contiennent les données que l'on souhaite stocker
  - *Exemples de fichiers* : une lettre tapée sous un éditeur de texte, un film, un programme, etc
- Répertoires : *(« dossiers » sous Windows)*
  - Pour ranger correctement les fichiers (« on ne mélange pas les torchons avec les serviettes »)
  - Peuvent contenir des fichiers ou d'autres dossiers
  - Contient toujours
    - Un raccourci vers lui-même : 
    - Un raccourci vers le répertoire « père » : 

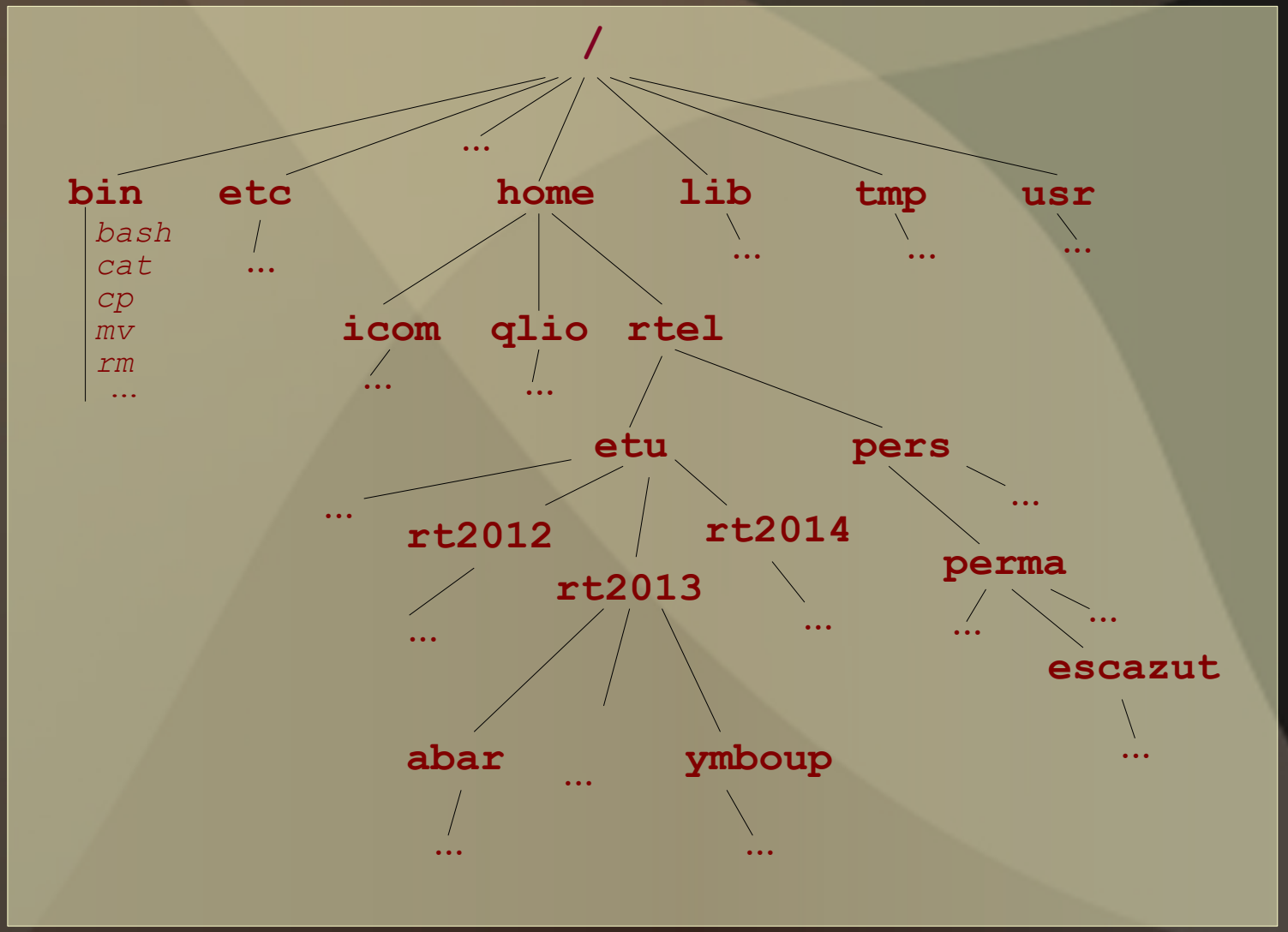
L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

- L'arborescence =
  - Tous les répertoires + tous les fichiers
  - Représenté comme un arbre …
    avec les racines en haut !
  - Le tout premier répertoire = la racine : /
  - Les feuilles = des fichiers



- L'arborescence Linux regroupe :
  - Le noyau du système
  - Les données de tous les utilisateurs :
    - Un répertoire personnel pour chacun
    - Différents niveaux d'autorisations

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits



L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

- Répertoire personnel :
  - Où on arrive dès que l'on se connecte
  - Les données qui y sont stockées nous appartiennent : on en fait ce que l'on veut !
  - Raccourci : `~`
- Pour désigner un fichier ou un répertoire :

```
<chemin dans l'arborescence>/<son nom>
```

  - `<son nom>` : le nom du répertoire ou du fichier (avec une extension si elle existe)

    Exemples : `home` `escazut` `cv.doc` `photo.jpg`

  - `<chemin>` : la liste des répertoires à suivre séparés par un `/` . Il peut être *absolu* ou *relatif*

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

- Chemin absolu :
  - Chemin exprimé à partir de la racine

    ➡ commence toujours par un /

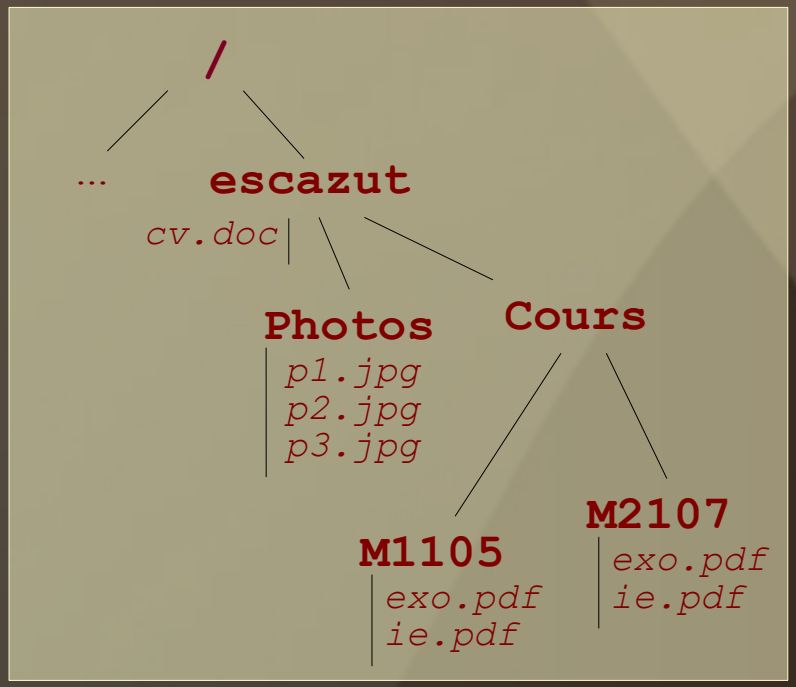
    Exemple : `/home/rtel/pers/perma/escazut/coucou.txt`
- Chemin relatif :
  - Chemin exprimé à partir de l'endroit où l'on se trouve

    ➡ commence toujours par ./ ou par ../

    Exemples : `./Cours/M1105/interro.pdf`
    `../../photo.jpg`
    `./zik.mp3`

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

Avez-vous compris ?



Seules deux de ces phrases sont exactes. Lesquelles ?

(a) `p2.jpg` est un *fichier*

(b) `p2.jpg` est un *répertoire*

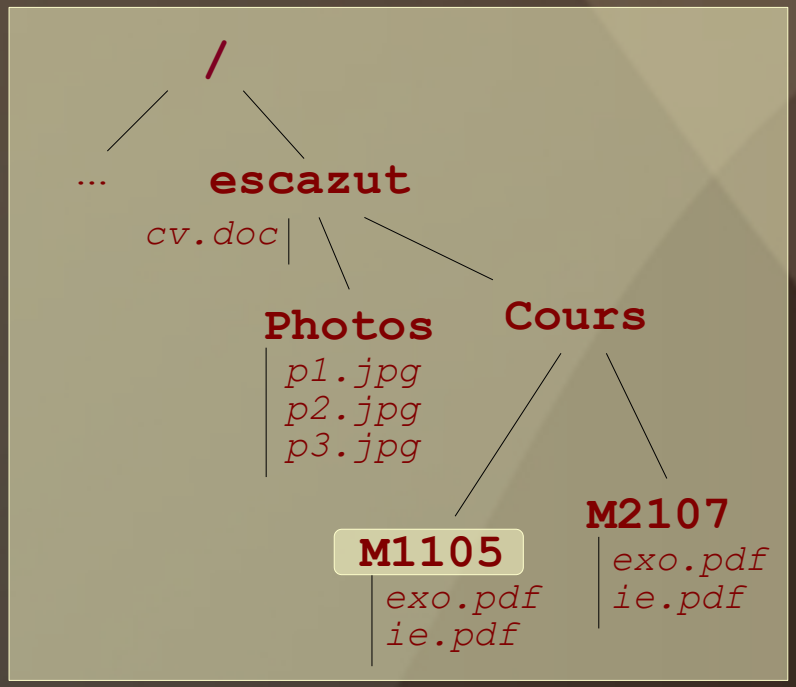
(c) `escazut` est un *fichier*

(d) `escazut` est un *répertoire*

L'arborescence

Manipuler les fichiers et les répertoires
Gestion des droits

Avez-vous compris ?



Le chemin *absolu* vers le répertoire `M1105` est :

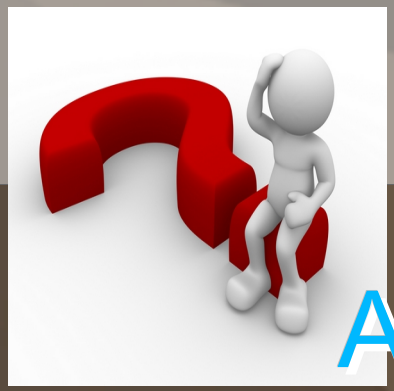
(a) `/M1105`

(b) `/escazut/Cours/M1105`

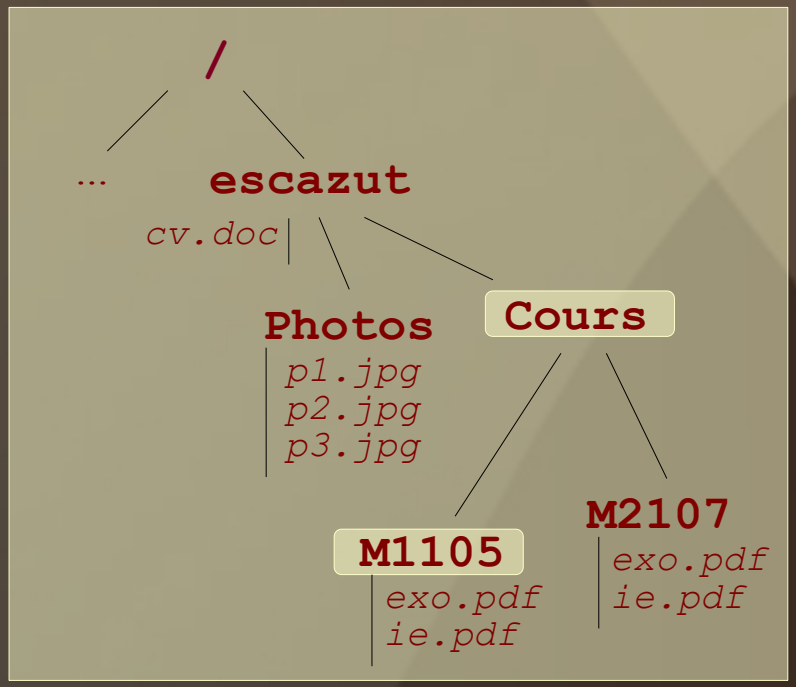
(c) `./M1105`

(d) `./escazut/Cours/M1105`

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits



Avez-vous compris ?



En partant du répertoire `Cours`, le chemin *relatif* vers le répertoire `M1105` est :

(a) `/M1105`

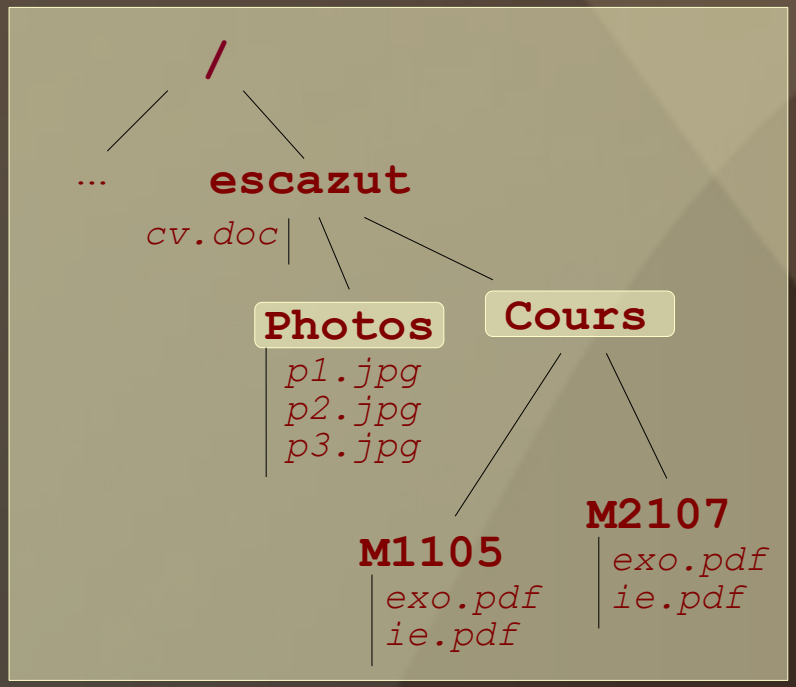
(b) `/escazut/Cours/M1105`

(c) `./M1105`

(d) `./escazut/Cours/M1105`

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

Avez-vous compris ?



En partant du répertoire `Cours`, le chemin *relatif* vers le répertoire `Photos` est :

(a) `/Photos`

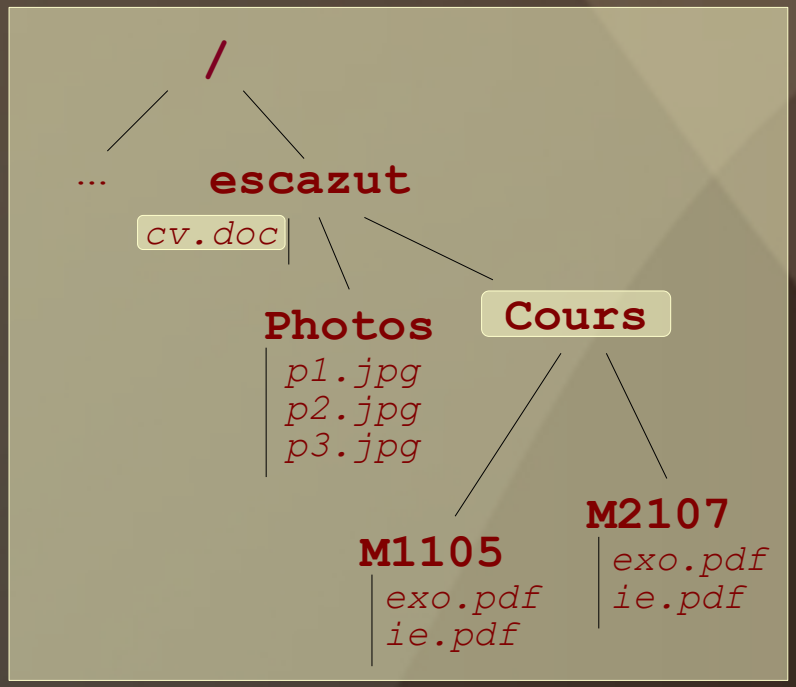
(b) `/escazut/Cours/Photos`

(c) `./Photos`

(d) `../Photos`

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

Avez-vous compris ?



En partant du répertoire `Cours`, le chemin *absolu* vers le fichier `cv.doc` est :

(a) `/cv.doc`

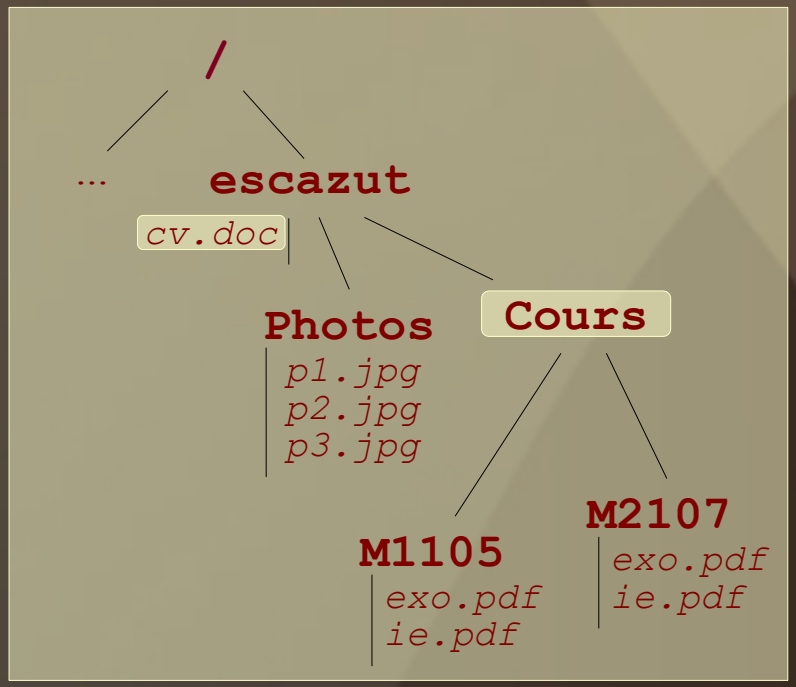
(b) `../cv.doc`

(c) `/escazut/cv.doc`

(d) `../escazut/cv.doc`

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

Avez-vous compris ?



En partant du répertoire `Cours`, le chemin *relatif* vers le fichier `cv.doc` est :

(a) `/cv.doc`

(b) `../cv.doc`

(c) `/escazut/cv.doc`

(d) `../escazut/cv.doc`

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

- Création d'un répertoire :

```
mkdir <rep_a_creer>
```

Exemple : `mkdir ./Cours/M2105`

- Copie d'un fichier ou d'un répertoire :

```
cp <fic_source> <destination>
```

```
cp -R <rep_source> <destination>
```

Exemple :
```
cp ../fichier1.pdf  ./fichier2.pdf
cp -R ./Photos  ./Svg
```

- Déplacer un fichier ou un répertoire :

```
mv <source> <destination>
```

Exemple : `mv ./fichier2.pdf  ..`

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

- Supprimer un fichier :

```
rm <fic_a_suppr>
```

Exemple : `rm ../fichier2.pdf`

- Supprimer un répertoire *vide* :

```
rmdir <rep_a_suppr>
```

Exemple : `rmdir ./Cours/M2105`

- Supprimer un répertoire et tout ce qu'il contient :

```
rm -r <rep_a_suppr>
```

Exemple : `rm -r ./Cours`

L'arborescence

Manipuler les fichiers et les répertoires

Gestion des droits

- Se déplacer dans l'arborescence :

```
cd <rep_ou_aller>
```

Exemple : `cd ./Cours/M1105`

- Afficher le nom du *répertoire courant* :

```
pwd
```

- Lister le contenu d'un *répertoire* :

```
ls <rep_a_lister>
```

Exemples :

```
ls ~          ls            ls ./Cours
ls -l         ls -a ./      ls -la ./Cours
```

L'arborescence

Manipuler les fichiers et les répertoires

Gestion des droits

- Lister les information concernant un *fichier* :

```
ls -l <fichier>
```

Exemples : `ls -l ./fichier1.pdf`

`ls -l ./Cours/M1105/interro.pdf`

- Afficher le contenu d'un *fichier* :

```
cat <fichier>
```

```
more <fichier>
```

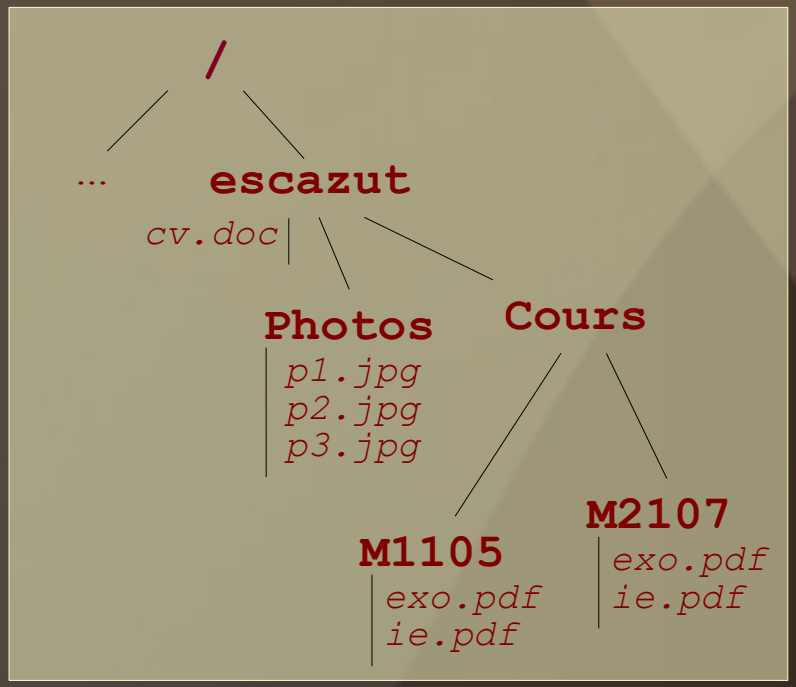
Exemples : `cat ./fic3.txt` `more ./fic3.txt`

L'arborescence

Manipuler les fichiers et les répertoires

Gestion des droits @

Avez-vous compris ?



Votre répertoire d'*entrée* est :

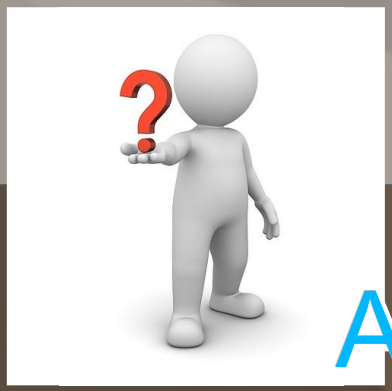
(a) Le répertoire `/`

(b) Le répertoire `.`

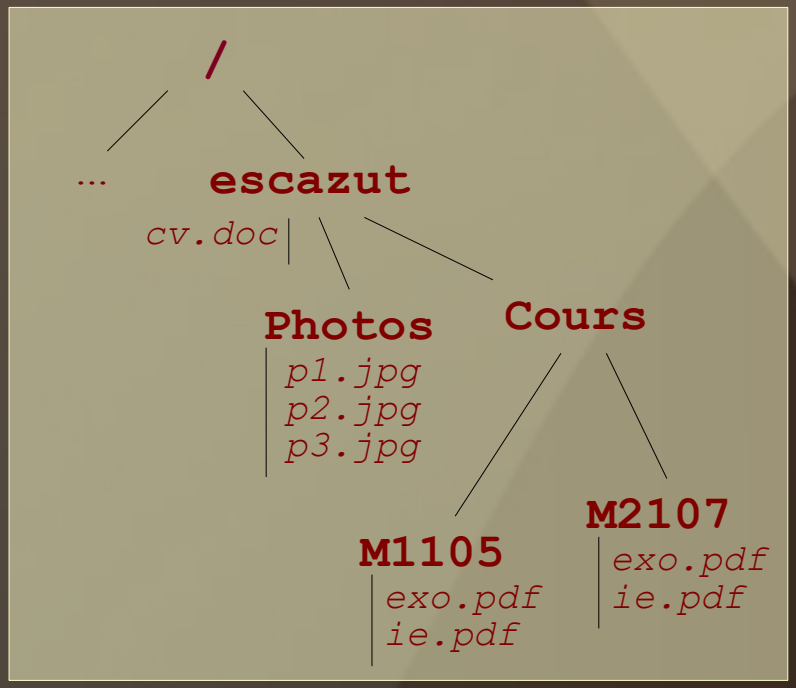
(c) Le répertoire `Cours`

(d) Le répertoire où j'arrive quand je me connecte

L'arborescence
- Manipuler les fichiers et les répertoires
Gestion des droits



Avez-vous compris ?

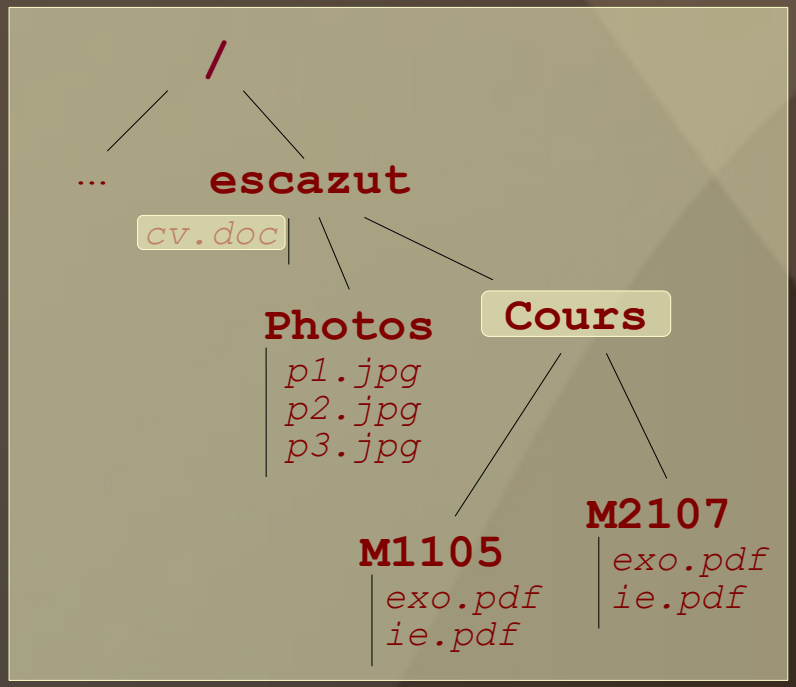


Votre répertoire *courant* est :

(a) Le répertoire `/`
(b) Le répertoire `.`
(c) Le répertoire `Cours`
(d) Le répertoire où vous êtes actuellement, c'est-à-dire le résultat de la commande `pwd`

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

Avez-vous compris ?



Votre répertoire *courant* est le répertoire `Cours`.
Les 2 commandes qui permettent de copier le fichier `cv.doc` dans votre répertoire *courant* sont :

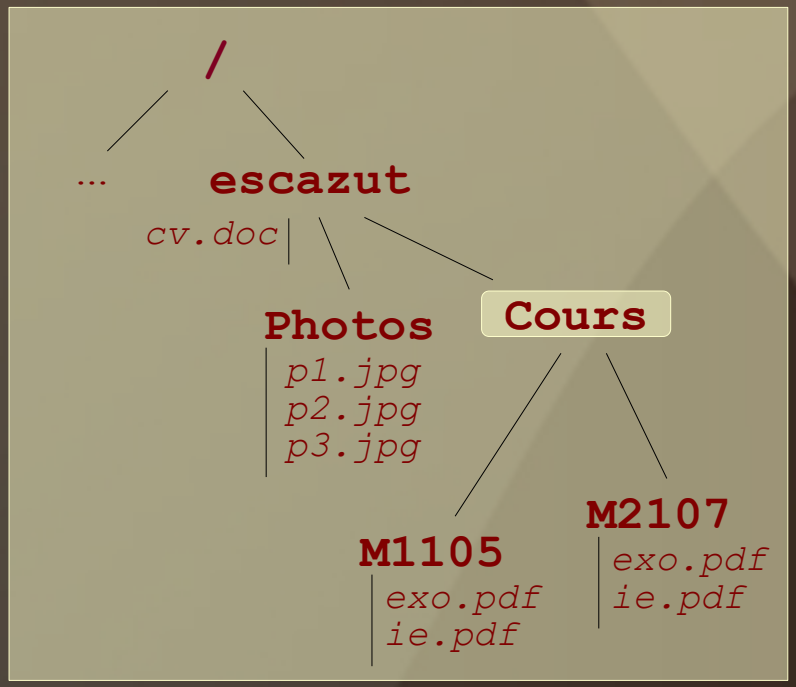
(a) `cp cv.doc Cours`
(b) `cp ./escazut/Cours cv.doc`
(c) `cp /escazut/cv.doc .`
(d) `cp ../cv.doc .`

L'arborescence

Manipuler les fichiers et les répertoires

Gestion des droits

Avez-vous compris ?



La commande pour lister le contenu de votre répertoire *d'entrée* est :

(a) `ls /`

(b) `ls ~`

(c) `ls pwd`

(d) `ls escazut`

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

- Linux est multi-utilisateurs :
  - Chaque utilisateur a un répertoire qui lui est dédié
  - Chaque utilisateur est responsable de ses données :
    - Respecter les règles (cf. charte de bon usage de l'informatique et du réseau)
    - Gère les droits sur ses fichiers et répertoires
  - Les fichiers et répertoires du système sont gérés par l'administrateur du serveur : `root`
  - L'administrateur du serveur possède tous les droits sur toute l'arborescence

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

- Changer les droits d'un fichier ou d'un répertoire:

```
chmod <droits> <fic_ou_rep>
```

- Qui a le droit de quoi ?
  - Les utilisateurs sont répartis en 3 catégories :

Le propriétaire	Les utilisateurs du groupe	Les autres utilisateurs
u	g	o

  - Il existe 3 types de droits :

Le droit de lecture	Le droit d'écriture	Le droit d'exécution
r	w	x

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

- `chmod` : exprimer les `<droits>` en mode *relatif* :
  - Partir des droits qui existent déjà
  - Ajouter ou enlever des doits à certains utilisateurs
  - Exprimer les droits en utilisant u g o et r w x :
    - Ajouter des droits : **`<categories>+<droits>`**
    - Enlever des droits : **`<categories>-<droits>`**
  - *Exemples* :

```
chmod  g+r  ./fic3.txt
chmod  ugo-wx  ../fic4.txt
chmod  a+wx  ../fic4.txt
```

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

- `chmod` : exprimer les `<droits>` en mode *absolu* :
  - Autoriser (ou pas) chacun des 3 droits pour chacune des 3 catégories d'utilisateurs : 9 informations
  - Utilisation d'un code binaire sur 9 bits :

    1 = donner le droit / 0 = ne pas donner le droit

u			g			o		
r	w	x	r	w	x	r	w	x
-	-	-	-	-	-	-	-	-

  - Chaque paquet de 3 bits est enfin exprimé sous forme décimale ➡ 3 chiffres entre 0 et 7 qui constituent le *masque des droits*
  - *Exemple* : `chmod 710 ./fic3.txt`

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

- Définir les droits initiaux des fichiers et répertoires :

  `umask <masque>`

  `<masque>` code inverse du masque du `chmod` :

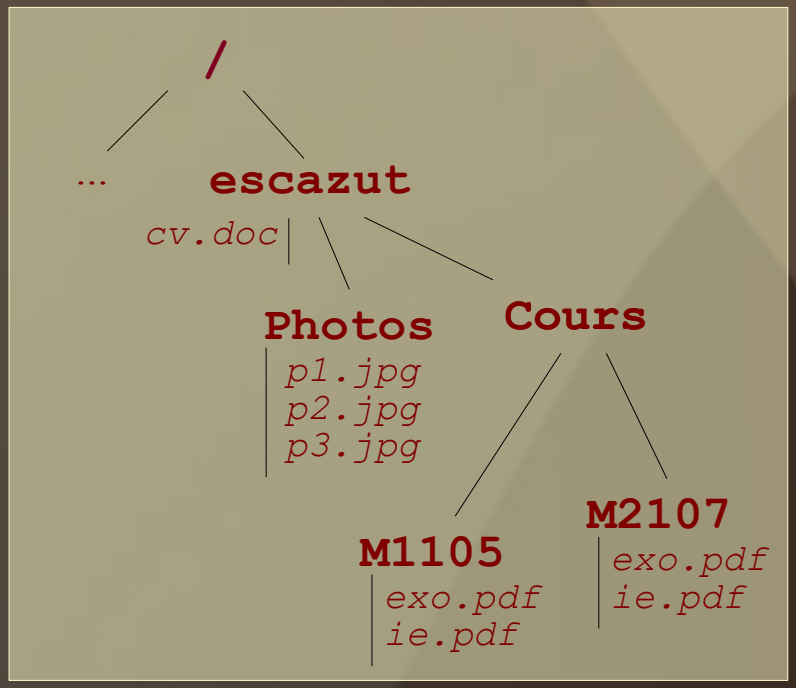
  0 = donner le droit / 1 = ne pas donner le droit

  - *Exemple* : `umask 067`

- Droits initiaux d'un *fichier* :
  - À sa création, et quel que soit le masque,
    un fichier n'aura jamais les droits d'**exécution**
- Droits d'exécution pour un *répertoire* :
  - Avoir le droit de s'y déplacer

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

Avez-vous compris ?



Vous n'êtes pas l'utilisateur `root`, vous avez le droit d'accéder :

(a) À toute l'arborescence sans aucune restriction

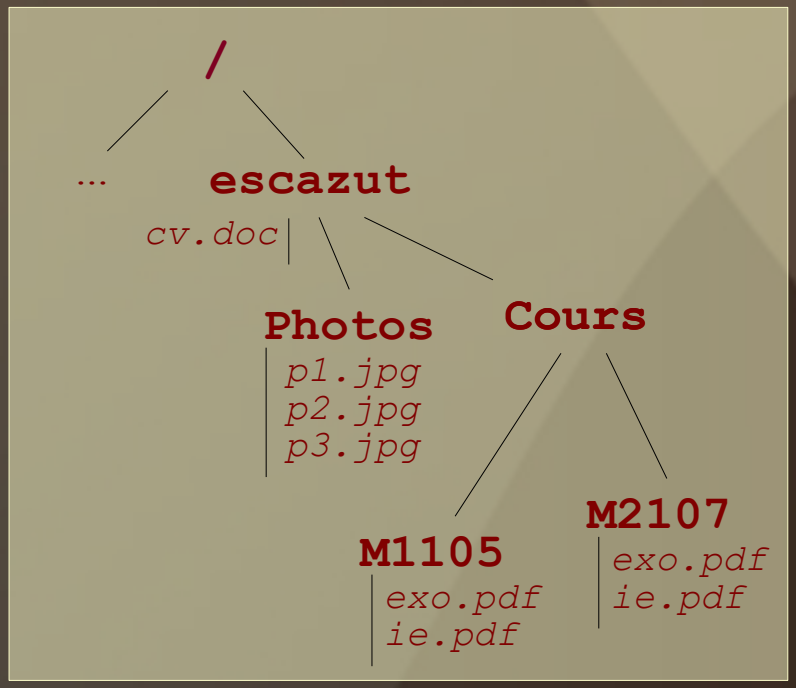
(b) Uniquement à mon répertoire d'entrée

(c) À rien tant que je ne mets pas les droits qu'il faut

(d) À mon répertoire d'entrée et aux répertoires des autres utilisateurs s'ils m'ont donné les droits

L'arborescence
Manipuler les fichiers et les répertoires
Gestion des droits

Avez-vous compris ?



Pour qu'un autre utilisateur ait accès à votre répertoire `Photos`, il faut :

(a) Qu'il soit l'utilisateur `root`

(b) Que son répertoire d'entrée soit placé à côté du mien

(c) Il ne pourra jamais y accéder, c'est interdit par Linux

(d) Que je lui ai donné les droits sur ce répertoire

3- Exécution des commandes LINUX

Les méta-caractères

Les redirections

Les processus

C. Escazut

- Méta-caractères :
  - Des caractères spéciaux qui permettent de spécifier plusieurs fichiers/répertoires ayant des similitudes dans leur nom.
  - *Exemple* : tous les fichiers et répertoires dont le nom commence par la lettre m et qui se termine soit par 1 soit par 2
  - `*` : une succession de 0, 1 ou plusieurs caractères
  - `?` : un et un seul caractère
  - `[ … ]` : un caractère choisi dans la liste entre crochets
  - `\` : enlève la spécificité au caractère qui suit

- Utilisation :
  - Taper un seule commande qui sera appliquée à plusieurs fichiers/répertoires
  - *Exemples :*

```
rm m*                rm m?
rm ../m*             rm m*[12]
rm m*m               rm m[12]
rm *                 rm m?[12]
rm ?                 rm \?
```

- Exécution d'une commande en 2 phases :

Commande = Évaluation + Exécution

① **Évaluation** : Ré-écriture de la commande en remplaçant tous les caractères spéciaux

② **Exécution** de la commande qui vient d'être ré-écrite

Exemple :

```
$ ls
m    maestro1    maman1    Moi2
$ rm m*[12]
```

`rm m*[12]` —① *Eval.*→ `rm maestro1` `rm maman1` —② *Exec.*→ Efface :

- `maestro1`
- `maman1`

Avez-vous compris ?

Un méta-caractère est un caractère :

(a) Qui exécute une commande Linux

(b) Qui n'existe pas dans l'alphabet classique mais que l'on utilise en Linux

(c) Qui en représente plusieurs autres

(d) Qui est plus fort que tous les autres

Avez-vous compris ?

Que se passe-t-il durant la phase d'évaluation d'une commande ?

(a) Linux évalue la commande pour savoir si elle ne contient pas d'erreur

(b) Linux ré-écrit la commande en remplaçant les méta-caractères

(c) Linux exécute la commande et affiche le résultat à l'écran

(d) Linux ne fait rien, il attend la phase d'exécution

Les méta-caractères
Les redirections
Les processus

- Interactions utilisateur – ordinateur :
  - Entrée standard : clavier
  - Sortie standard : écran
- Rediriger la sortie standard :
  - Stocker le résultat d'une commande dans un fichier
  - Pas d'affichage à l'écran
  - Syntaxes :

```
<cmde>  >  <nom_fic>
<cmde>  >>  <nom_fic>
```

Les méta-caractères
Les redirections
Les processus

- Rediriger l'entrée standard :
  - La commande va chercher les informations dont elle a besoin dans un fichier
  - L'utilisateur ne tape rien au clavier
  - Syntaxe :

<cmde> < <nom_fic>

Exemples :

```
$ ls .. > contenu
$ ls >> contenu
$ mail cathy.escazut@unice.fr < contenu
```

Les méta-caractères
Les redirections
Les processus

- Enchainer les commandes :
  - La sortie d'une commande sert d'entrée à une autre commande
  - Solution 1 : passer par un fichier intermédiaire
  - Solution 2 : pas de fichier intermédiaire

```
<cmde1>  |  <cmde2>
```

Exemples :

```
$ ls > tmp ; more tmp ; rm tmp
$ ls | more
$ who | wc -l
```

Avez-vous compris ?

Rediriger la sortie standard signifie que le résultat de la commande :

(a) s'affichera sur un autre écran

(b) sortira sur une imprimante

(c) sera rangé dans un fichier à la place d'être affiché à l'écran

(d) sera affiché sur l'écran

Avez-vous compris ?

Les tubes (caractère |) permettent de :

(a) d'accélérer l'exécution des commandes

(b) je ne sais pas, j'ai rien compris

(c) de ranger les résultats en attendant d'en avoir besoin plus tard

(d) de relier des commandes entre elles

Les méta-caractères
Les redirections
▸ Les processus

- Un processus =
  - Un programme qui tourne
  - Toujours exécuté par un *processus père*
  - Représenté par un numéro : `pid`
- Avoir la liste des processus qui tournent : `ps`
  - Plusieurs options
  - Quelques informations utiles données par ps :
    - `UID` : identifiant de l'utilisateur ayant lancé le processus
    - `PID` : identifiant du processus
    - `PPID` : identifiant du processus père

Les méta-caractères
Les redirections
▸ Les processus

- Exécuter un processus en « avant-plan » :
  Taper son nom et les paramètres éventuels :
  - Attribution automatique d'un `pid` (visible avec `ps`)
  - Attendre la fin du processus pour récupérer la main
- Exécuter un processus en « arrière-plan » :
  Reprendre la main de suite (pas d'attente)
  - Lancer un processus en arrière-plan : `<cmde> &`
  - Mettre un processus en pause : `<ctrl>-Z`
  - Passer un processus de l'avant à l'arrière-plan : `bg`
  - Passer un processus de l'arrière à l'avant-plan : `fg`
  - Liste des processus en arrière-plan : `jobs`

Les méta-caractères
Les redirections
▸ Les processus

- Stopper un processus :
  - Solution 1 : `<Ctrl> C`
    - Tue uniquement le processus en *avant-plan*
  - Solution 2 : `kill -9 <pid>`
    - Connaître le pid du processus à « tuer »
    - Valable pour tous les processus
- Un processus important : `bash`
  - Programme qui contient les commande linux
  - `~/.bashrc` : fichier de configuration

Avez-vous compris ?

Un processus est :

(a) un programme qui s'exécute
(b) un des composants d'un ordinateur
(c) un programme qui tue les autres programmes
(d) une méta-commande

4- Les expressions régulières

Introduction

Les caractères spéciaux

Les commandes `grep` et `sed`

C. Escazut

- Expressions régulières :
  - Pour analyser et traiter du texte de façon très rapide
    - Vérification de données
    - Mise en forme d'un texte
    - Recherche d'informations précises dans un fichier
  - Utilisées par de nombreux langages
  - Expressions basées sur des « *motifs* » : un vrai langage qui utilise des *caractères spéciaux*
  - Ne pas confondre les expressions régulières et les méta-caractères du shell :

Les expressions régulières ne sont utilisées que par certaines commandes du shell

Introduction
Les caractères spéciaux
Les commandes `grep` et `sed`

- Exprimer une position :
  - `^` : début d'une ligne
  - `$` : fin d'une ligne
  - *Exemples* : `^1` `9$` `^a$` `^$`
- Exprimer des caractères :
  - `<car>` : le caractère `<car>`
  - `.` : un caractère quelconque
  - `\<car>` : banalise le caractère spécial `<car>`

  Exemples : `^abcd` `zzz$` `^z.$` `^ba.b$` `\.` `^b\..b$`

Introduction
Les caractères spéciaux
Les commandes `grep` et `sed`

- Exprimer la répétition d'un caractère :
  - **`<car>*`** : le caractère `<car>` est répété 0, 1 ou plusieurs fois
  - **`<car>?`** : le caractère `<car>` est répété 0 ou 1 fois
  - **`<car>+`** : le caractère `<car>` est répété 1 ou plusieurs fois

Exemples : `ab*c` `^a*$` `^b.*b$`

`^ab?a$` `^ab*a$` `^ab+a$`

Introduction
Les caractères spéciaux
Les commandes `grep` et `sed`

- Choix dans une liste :
  - **`[<listeCar>]`** : un seul caractère parmi ceux présents dans la liste `<listeCar>`
  - **`[^<listeCar>]`** : un seul caractère qui n'est pas dans la liste de caractères `<listeCar>`
  - **`[<car1>-<car2>]`** : un caractère pris dans l'intervalle entre `<car1>` et `<car2>` (chiffres, minuscules ou majuscules)

  Exemples :

```
[aceg]          ^[ab]          ^[^ab]*$
[a-z][A-Z]      1[0-9]         [0-9]+
```

Avez-vous compris ?

L'expression régulière `^...[zZ]` représente une ligne qui :

(a) Commence par 3 chiffres suivis de zZ

(b) Possède la lettre z (minuscule ou majuscule) en 4ème position

(c) Contient une lettre quelconque après 3 points

(d) Est composée de 7 caractères et se terminant par zZ

Avez-vous compris ?

La commande `rm ^[a-z]*$` essaie d'effacer

(a) le fichier dont le nom est `^[a-z]*$`

(b) les fichiers dont le nom est composé uniquement de lettres minuscules

(c) rien car il y a une erreur

(d) les fichiers dont le nom commence par `^a` ou `^-` ou `^z` et qui se termine par un `$`

Introduction
Les caractères spéciaux
Les commandes `grep` et `sed`

Avez-vous compris ?

Quelle expression régulière repère toutes les lignes contenant un numéro de portable (ils commencent par 06 ou 07) ?

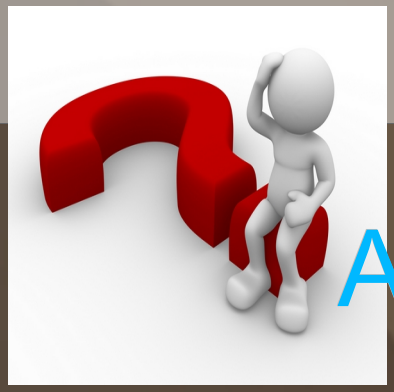
(a) `*[06-07]*`

(b) `$0[67].*`

(c) `0[67][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]`

(d) `^06|07*$`

Introduction
Les caractères spéciaux
Les commandes `grep` et `sed`



Avez-vous compris ?

Quelle expression régulière repère toutes les lignes qui ne contiennent pas de caractère a ?

(a) `\a*`

(b) `^[\a]*$`

(c) `^[^a]*`

(d) `^[^a]*$`

Introduction
Les caractères spéciaux
Les commandes `grep` et `sed`

Avez-vous compris ?

Quelle expression régulière repère toutes les lignes qui contiennent le *mot* velo1 ou velo10 ou velo11 ou … velo19 ?

(a) `␣velo1[0-9]?␣`
(b) `␣velo1[0-9]*␣`
(c) `␣velo1[0-9]+␣`
(d) `velo1[0-9].`

Introduction
Les caractères spéciaux
Les commandes `grep` et `sed`

- Expressions régulières *étendues* :
  - **`(<exp_reg>)?`** : la chaîne correspondant à `<exp_reg>` sera répété 0 ou 1 fois
  - **`(<exp_reg>)+`** : la chaîne correspondant à `<exp_reg>` sera répété au moins 1 fois
  - **`(<exp_reg>){<a>,<b>}`** : la chaîne correspondant à `<exp_reg>` sera répétée entre <a> et <b> fois
  - **`(<exp_reg1>)|(<exp_reg2>)`** : la chaîne correspondant à `<exp_reg1>` ou à `<exp_reg2>`

Exemples : `(abc)|(def)` `(ab)?` `(a.)+` `(ab){5,9}` `(a.){5}`

Introduction
Les caractères spéciaux
Les commandes `grep` et `sed`

- La commande `grep` :
  - Afficher toutes les lignes d'un fichier qui contiennent un motif recherché

```
grep '<exp_reg>' <fic>
```

  - Quelques options
    - `-v` : afficher toutes les lignes qui ne contiennent pas le motif recherché
    - `-E` : utiliser des expressions régulières étendues (`egrep`)

Exemples :

```
grep '^a.*z$' fic1.txt
grep 'root' fic2.txt > root.txt
grep '^$' fic3.txt |  wc -l
```

Introduction
Les caractères spéciaux
Les commandes `grep` et `sed`

- La commande `sed` :
  - Lit une à une les lignes d'un fichier
  - Leur applique un traitement (substitution de chaînes)
  - Affiche le résultat sur la sortie standard
  - Ne modifie pas le fichier

Introduction
Les caractères spéciaux
Les commandes grep et sed

- La commande sed (suite) :
  - Syntaxe :

sed -e '<cmde_sed>' <fic>

Avec <cmde_sed> qui vaut :

<ligDeb>,<ligFin>s/<quoiRemplacer>/<parQuoi>/

Exemples :

```
sed -e '5,8s/[wW]indows/LINUX !/' f1.txt
sed -e '5,8s/[wW]indows/LINUX !/3' f2.txt
sed -e '5,8s/[wW]indows/LINUX !/g' f3.txt
```

Introduction
Les caractères spéciaux
Les commandes `grep` et `sed`

- La commande `sed` (suite) :
  - Mémoriser des sous-chaînes pour les réutiliser

`\(<motif>\)` `\1 \2 …`

Exemples :

```
sed -e 's /\([wW]indows\)is good\([lL]inux\)is bad/
          \2 = good \1 = bad/' f4.txt  > f4.txt
```

```
sed -e 's/^\(.\).*\(.\)$/\1...\2/' f5.txt
```

5- Les variables

Principes de base

Retour sur la phase d'évaluation

Variables locales, globales et prédéfinies

C. Escazut

Principes de base
Retour sur la phase d'évaluation
Variables locales, globales et prédéfinies

- Variable :
  - Un moyen de donner un nom à une valeur qui peut changer et que l'on veut pouvoir utiliser à tout moment
  - Métaphore : une variable est une boîte nommée dans laquelle on range une valeur que l'on voudra utiliser plus tard
- Caractéristiques d'une variable :
  - Un nom (commence par une lettre) : `<nomVar>`
  - Un contenu (nom précédé d'un $) : `$<nomVar>`

  Exemple : `age` ⟶ `$age`

Principes de base
Retour sur la phase d'évaluation
Variables locales, globales et prédéfinies

- Stocker une valeur dans une variable :
  - Affectation : **<nomVar>=<valeur>**

    Exemples : `x=5` `y=hello` `z='bye bye'`
  - Valeur donnée par l'utilisateur : **read <nomVar>**
    - La commande s'interrompt et attend que l'utilisateur donne une valeur sur l'entrée standard
    - La valeur est stockée dans la variable
    - Possibilité de lire plusieurs valeurs de variables :

      read <nomVar1> <nomVar2> ...

    Exemples : `read nom` `read val1 val2`

Principes de base
Retour sur la phase d'évaluation
Variables locales, globales et prédéfinies

- Afficher : la commande `echo`

`echo <unMessage>`

`echo <contenuVar>`

Exemples :

```
$ age=18
$ echo $age
18
$ echo date de naissance
date de naissanace
$ read jour mois annee
21 12 1994
$ echo né le $jour-$mois-$annee
né le 21-12-1994
```

Principes de base

Retour sur la phase d'évaluation

Variables locales, globales et prédéfinies

- Rappel : Avant d'être exécutée une commande est évaluée
- Évaluation des contenus de variables :
  - À l'évaluation `$<nomVar>` remplacé par le contenu de `<nomVar>`

Exemple :

```
echo né le $jour-$mois-$annee
```

Eval.

```
echo né le 21-12-1994
```

Exec.

Affiche : `né le 21-12-1994`

Principes de base

Retour sur la phase d'évaluation

Variables locales, globales et prédéfinies

- Autres méta-caractères :
  - **" … $<nomVar> … "**
    ➡ $<nomVar> est évalué
  - **' … $<nomVar> … '**
    ➡ $<nomVar> n'est pas évalué
  - **`<cmdShell>`**
    ➡ <cmdShell> exécutée à l'évaluation

Exemples :

```
y=Bonjour
echo "$y à tous"
echo '$y à tous'
echo je suis en `pwd`
```

Principes de base
Retour sur la phase d'évaluation
Variables locales, globales et prédéfinies

- Variables locales :
  - En général, variables connues uniquement dans le shell où elles ont été créées : Variables Locales
  - Liste des variables locales : `set`
- Variables globales :
  - Certaines variables sont connues dans tous les shell
  - Liste des variables globales : `env`

Principes de base
Retour sur la phase d'évaluation
Variables locales, globales et prédéfinies

- Transformer une variable locale en globale :

```
export <nomVarLocale>
```

Exemple :

```
$ age=18
$ set | grep age=18
age=18
$ env | grep age=18
$ export age
$ env | grep age=18
age=18
```

Principes de base
Retour sur la phase d'évaluation
Variables locales, globales et prédéfinies

- Quelques variables prédéfinies du shell (locales !) :
  - `PS1` : contient le prompt
  - `HOME` : contient le répertoire d'entrée
  - `IFS` : séparateur de mots
  - `1`, `2`, `3` : créées automatiquement
  - `#` : nombre de variables créées automatiquement
  - `?` : compte-rendu d'exécution

```
$ echo $HOME
/home/retl/pers/perma/escazut
$ PS1="Oui chef ? > "
Oui chef ? > ls
```

Principes de base
Retour sur la phase d'évaluation
▸ Variables locales, globales et prédéfinies

- Commande `set` :
  - Découpe une *chaîne* en fonction du caractère espace
  - Range chaque valeur dans les variables **1**, **2**, …
  - Met à jour la variable **#**

```
$ set toto tutu titi
$ echo "Les 3 morceaux = $1 - $2 - $3"
Les 3 morceaux = toto - tutu - titi
$ msg='bonjour à tout le monde'
$ set $msg
$ echo "il y a $# mots dans msg"
Il y a 5 mots dans msg
```

Principes de base
Retour sur la phase d'évaluation
Variables locales, globales et prédéfinies

- Dernier retour sur l'évaluation :

  Différence d'évaluation entre `$x` et `"$x"`

  - À l'évaluation, `$x` est remplacé par le contenu de la variable `x` et les caractères de l'IFS sont remplacés par un espace
  - À l'évaluation, `"$x"` est remplacé par le contenu de la variable `x`, les caractères de l'IFS restent inchangés

```
$ IFS='-'
$ x=aa-bb-cc
$ echo $x
aa bb cc
$ echo "$x"
aa-bb-cc
```

6- Les scripts Shell

Introduction

Le langage de scripts

Variables inhérentes à un script

C. Escazut

Introduction
Le langage de scripts
Variables inhérentes à un script

- Un script =
  - Un fichier texte qui contient une succession de commandes qui seront exécutées les unes après les autres
  - Permet d'automatiser des tâches répétitives
  - Existence de plusieurs langages de scripts
- Un script shell :
  - Contient des commandes shell, manipule des variables, utilise des conditions et des boucles, ...
  - Commence toujours par : `#!/bin/bash`

  (indique le shell qui exécutera les commandes)

Introduction
Le langage de scripts
Variables inhérentes à un script

- 2 façons d'exécuter un script shell :
  - Dans un processus fils :
    - `bash <nom_script>`
    - `./<nom_script>` (droit d'exécution)

    → Création d'un processus temporaire qui se terminera à la fin de l'exécution du script

  - Dans le processus courant :

    `source <nom_script>`

    → Pas création de nouveau processus

Introduction
Le langage de scripts
Variables inhérentes à un script

Exemple 1 :

```
$ cat exemple1
#!/bin/bash

echo liste des process qui tournent :
ps lf
echo … et voilà !

$ source exemple1
```

Introduction
Le langage de scripts
Variables inhérentes à un script

- Exprimer une condition

Si < condition >
Alors
    < actionsAlors >
Sinon
    < actionsSinon >
Finsi

```
if <condition>
then
   <cmdesAlors>
else
   <cmdesSinon>
fi
```

La partie **else** est facultative

Introduction

Le langage de scripts

Variables inhérentes à un script

- Exprimer des itérations :

TantQue < condition >
Faire
    < actions >
FinTantQue

```
while <condition>
do
   <cmdes>
done
```

Pour < var > dans < liste >
Faire
< actions >
FinPour

```
for <var> in <liste>
do
   <cmdes>
done
```

Introduction
Le langage de scripts
Variables inhérentes à un script

Exemple 2 :

```
$ cat exemple2
#!/bin/bash

for i in Pommes Bananes Figues
do
    echo Des $i
done

$ bash exemple2
```

Introduction
Le langage de scripts
Variables inhérentes à un script

- La commande **`test`** =
  - Utilisée dans **`if`** et **`while`** (parties `<condition>`)
  - Pour faire des tests sur les fichiers, les chaînes de caractères, les nombres, les variables
  - Comme toutes les commandes du shell la commande **`test`** retourne un code :
    - 0 (vrai) si exécution OK
    - ≠0 (faux) si exécution KO
  - 2 Syntaxes au choix :

```
test <condition>
```

```
[  <condition>  ]
```

Introduction
Le langage de scripts
Variables inhérentes à un script

- La commande **`test`** =
  - Utilisée dans **`if`** et **`while`** (parties `<condition>`)
  - Pour faire des tests sur les fichiers, les chaînes de caractères, les nombres, les variables
  - Comme toutes les commandes du shell la commande **`test`** retourne un code :
    - 0 (vrai) si exécution OK
    - ≠0 (faux) si exécution KO
  - 2 Syntaxes au choix :

```
test <condition>
```

```
[  <condition>  ]
```

Introduction
Le langage de scripts
Variables inhérentes à un script

- **test** et les fichiers/répertoires :

`test -d <nom>`	<nom> est un répertoire qui existe ?
`test -f <nom>`	<nom> est un fichier qui existe ?
`test -r <nom>`	<nom> est en lecture ?
`test -w <nom>`	<nom> est en écriture ?
`test -x <nom>`	<nom> est en exécution ?

  - D'autres options : **-ot**, **-s**, …

- **test** et les chaînes de caractères/variables :

`test -z <ch>`	<ch> est vide ?
`test <ch1>=<ch2>`	<ch1> et <ch2> identiques ?
`test <ch1>!=<ch2>`	<ch1> et <ch2> différentes?

Introduction
Le langage de scripts
Variables inhérentes à un script

- Exemple 3 :

```
$ cat exemple3
#!/bin/bash

if test -f  toto
then
    echo le fichier toto existe
else
    echo pas de fichier toto
fi

$ ./exemple3
```

Introduction
Le langage de scripts
Variables inhérentes à un script

- **`test`** et les nombres :

`[ <n1> -eq <n2> ]`	<n1> = <n2> ?
`[ <n1> -ne <n2> ]`	<n1> ≠ <n2> ?
`[ <n1> -lt <n2> ]`	<n1> < <n2> ?
`[ <n1> -le <n2> ]`	<n1> ≤ <n2> ?
`[ <n1> -gt <n2> ]`	<n1> > <n2> ?
`[ <n1> -ge <n2> ]`	<n1> ≥ <n2> ?

Introduction
Le langage de scripts
Variables inhérentes à un script

- Exécution d'une portion de script dans *un clone du shell* : utilisation de parenthèses

Exemple 4 :

```
$ cat exemple4
#!/bin/bash
pwd
x=aaa ; echo $x
(
    cd .. ; pwd ; echo $x
    x=bbb ; echo $x
)
pwd
echo $x
```

Introduction
Le langage de scripts
Variables inhérentes à un script

- Variables toujours présentes et connues dans un script :
  - Nom du script : `0`
  - Paramètres passés lors de l'appel du scripts sont rangés dans les variables `1, 2 …`
  - Liste des paramètres **1**, **2**, ... : `*`
  - Nombre des paramètres : `#`
  - Code retourné par une commande : `?`

→ Pour accéder aux valeurs :

$0 $1 … $* $# $?

Introduction
Le langage de scripts
Variables inhérentes à un script

Exemple 5 :

```
$ cat exemple5
#!/bin/bash
echo nom du script = '$0'= $0
echo parametre 1 = '$1'= $1
echo parametre 2 = '$2'= $2
echo parametre 3 = '$3'= $3
echo nb de parametres = '$#'= $#

$ bash exemple5 10 hello 30
```

Introduction
Le langage de scripts
Variables inhérentes à un script

Exemple 6 :

```
$ cat exemple6
#!/bin/bash
echo les $# parametres sont :
for i in $*
do
  echo $i
done
$ ./exemple6 toto va a la plage
```

Introduction
Le langage de scripts
Variables inhérentes à un script

- Commande **shift** décale les valeurs :

1 ← **$2**, **2** ← **$3**, ... , **#** ← **$#** - 1

Exemple 7 :

```
$ cat exemple7
#!/bin/bash
echo '$0'= $0, '$#'= $#, '$1'=$1, '$2'=$2
shift
echo '$0'= $0, '$#'= $#, '$1'=$1, '$2'=$2

$ bash exemple7 aaa bbb
```

Introduction
Le langage de scripts
Variables inhérentes à un script

Exemple 8 :

```
$ cat exemple8
#!/bin/bash
echo '$0'= $0, '$#'= $#, '$1'=$1, '$2'=$2
set aaa  bbb
echo '$0'= $0, '$#'= $#, '$1'=$1, '$2'=$2

$ bash exemple8
```

Introduction
Le langage de scripts
Variables inhérentes à un script

Exemple 9 :

```
$ cat exemple9
#!/bin/bash
echo '$0'= $0, '$#'= $#
echo 'une phrase ? '
read clavier
set $clavier
echo '$0'= $0, '$#'= $#
for i in $*
do
  echo $i
done

$ bash exemple9
```