TD1

Linux : les bases

Patrick FULCONIS

Présentation de l'UE

- **thèmes et aspects pratiques**
  - But : acquérir l'autonomie nécessaire à la compréhension et à la résolution des problèmes liés à l'environnement de programmation et au système d'exploitation, et introduction aux modèles mathématiques et informatiques qui structurent le comportement d'un système.
  - Programme :
    - éléments de système : interpréteur, système de fichiers, programmation shell
    - programmation en C : codage de l'information, bibliothèques standard (notamment I/O), mémoire (notamment les pointeurs et tableaux)
    - outils : gcc, make, gdb
    - introduction aux automates
    - projet : utilisation des notions précédentes pour la réalisation d'une partie d'un interpréteur de commandes simple

Présentation de l'UE

- **thèmes et aspects pratiques**
  - Organisation : 4 créneaux par semaine, dont 2 TPs sur machine l'un à la suite de l'autre. Pour ces derniers, le premier est encadré, le second ne l'est pas (il permet de terminer le travail demandé).
  - TD/CTD : préparation des TPs à venir, et on revient sur ceux qui ont été faits
  - TPs (en binôme) : exercices. Certaines notions sont vues pour la première fois en TP. Rédaction d'un compte rendu à rendre à la séance de TD suivante. Ces comptes-rendus sont corrigés mais pas notés.

  - Evaluation :
    - CC1 : un DS (30%)
    - CC2 : des petites interros (20mn) les semaines 3, 5, 7, 9, 11 (20%)
    - un examen terminal (50%). (rattrapage uniquement exam)

Linux

Unix/Linux différencie les majuscules des minuscu

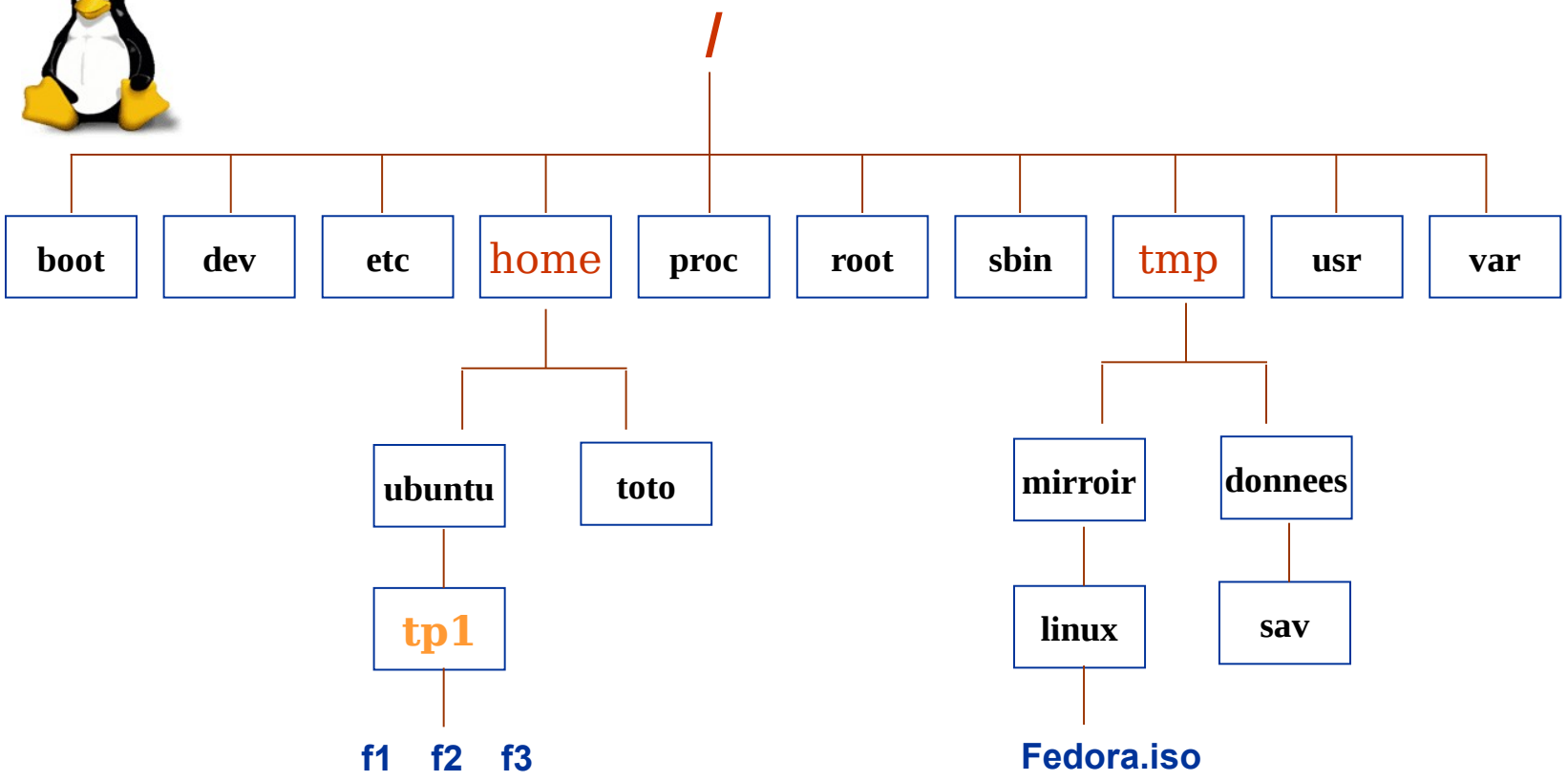
Unix n’aime pas les espaces **et** caractères spéciaux **dans les noms de fichiers, ni les** accents

L’espace sert de séparateur entre commandes/options/arguments

Arborescence (Chemin/path)

- **Tout répertoire/fichier peut être identifié de deux manières différentes**
  - Par son chemin (référence) absolu
    - Liste des répertoires depuis la racine (/) jusqu'au fichier recherché.
  - Par son chemin (référence) relatif
    - Liste des répertoires depuis le répertoire courant (là ou l'on se trouve) jusqu'au fichiers recherché.
  - On appellera *nom* d'un fichier ou d'un répertoire son chemin relatif depuis le répertoire où il se trouve
    - **c'est-à-dire la fin de tous les chemins qui le désignent**

Arborescence (Chemin/path)



Répertoire courant : /home
Nom répertoire de destination : tp1

Chemin absolu : /home/ubuntu/tp1

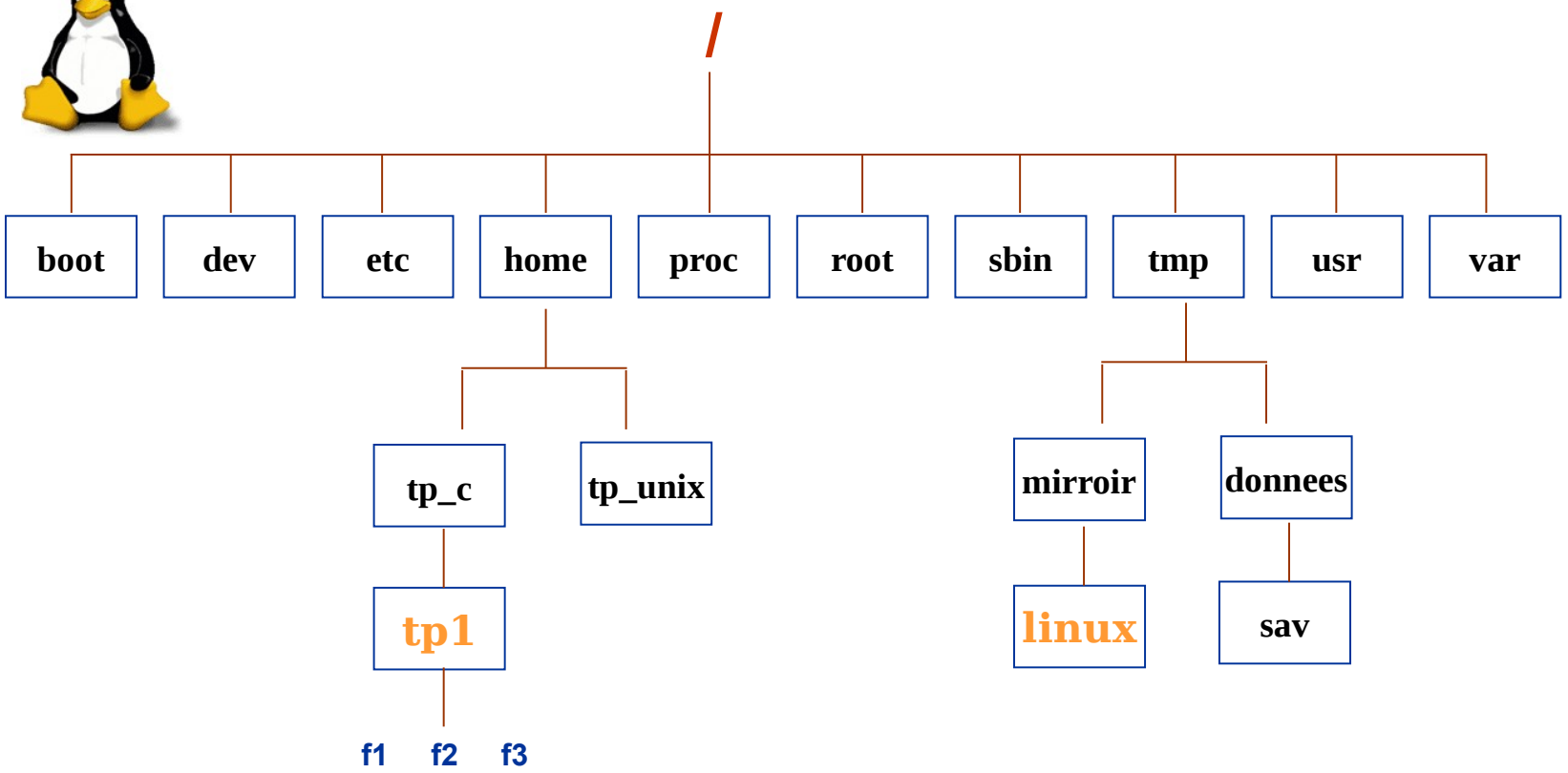
Chemin relatif : ubuntu/tp1

Répertoire courant : /tmp
Nom fichier de destination : **Fedora.iso**

Chemin absolu : /tmp/mirroir/linux/Fedora.i

Chemin relatif : mirroir/linux/Fedora.iso

Arborescence (Chemin/path)



Répertoire courant : **tp1**
Nom répertoire de destination : **linux**

Chemin absolu : /tmp/mirroir/linux

Chemin relatif : ../../../tmp/mirroir/linux

Arborescence (Chemin/path)

- **Relation père/fils**
- **Répertoires :**

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

- **Remarques :**
  - Il existe de nombreux chemins relatifs possibles pour un fichier
    - En général on choisit le plus cours
    - Mais les aller-retours ne sont pas interdit
  - **En général, on utilise le chemin le plus pratique : relatif quand on est proche, absolu quand on est loin dans l'arbre.**

Les commandes

- **Syntaxe générale**

  commande [-options] [arguments] arguments

  la commande peut être suivie d'options ou d'arguments (paramètres) (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

Les commandes

- **Sur les répertoires :**

pwd : affiche le répertoire courant (***p****rint* ***w****orking* ***d****irectory*)

cd : change de répertoire courant (***c****hange* ***d****irectory*)

cd .. remonte au répertoire père

cd / va au répertoire racine

cd va au répertoire home de l'utilisateur

cd - va au répertoire précédent

mkdir : crée un nouveau répertoire (**m**a**k**e **dir**ectory)

mkdir toto crée le répertoire toto

rmdir : efface un répertoire vide (**r**e**m**ove **dir**ectory)

rmdir toto supprime le répertoire toto

Les commandes

- **Sur les fichiers :**

ls liste les fichiers et répertoires (***l****i****s****t***)

cp copie des fichiers et répertoires (***c****o****p****y***)

rm supprime des fichiers et répertoires (***r****e****m****ove***)

mv déplace et/ou renomme un fichier ou un répertoires (***m****o****v****e***)

touch crée un fichier vide (ou change la date d'un fichier existant)

Manipulation de fichiers

- **Visualisation de fichiers texte**

Utilisation des commandes (en mode texte) :

cat, more et less

cat (concaténer) : permet d'afficher, de créer, copier et de concaténer des fichiers.

Syntaxe : cat nom_fichier

Manipulation de fichiers

Visualisation de fichier texte (suite)

more (ou less): affiche un fichier page par page ( )

Syntaxe : more nom_fichier

Avant d'utiliser les commandes d'affichage, il est nécessaire de
connaître le type de fichier(texte, binaire, exécutable).

La commande file permet d'identifier le type d'un fichier.

Syntaxe : file nom_fichier

Éditeurs mode texte : vi, vim, **emacs, gedit** …

Manipulation de fichiers

- **Copie, déplacement et effacement de fichiers**

Copie de fichiers

cp (copy) permet :

- de copier des fichiers
- des répertoires avec l'option –R ou -r

Syntaxe :

cp *source destination*

source = 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 rep toto

Manipulation de fichiers

- **Copie, déplacement et effacement de fichiers (suite)**

Attention :

source = fichiers à copier/déplacer
source peut être des fichiers/répertoires

destination = où on les copie/déplace
destination peut exister ou non, être un répertoire ou fichier

Exemples :

cp *fichier1 fichier2*
cp *fichier1 /tmp/fichier2*
cp *fichier1 /tmp/*
cp */tmp/fichier1* **.**
cp *fichier1 tmp* **(***tmp* **fichier ou repertoire)**
cp –R *rep1 rep2* **(***rep2* **existe ou non)**
cp –R */tmp/rep1 /tmp2/rep2*

Manipulation de fichiers

- **Conventions**

  quand il y a *fich* dans un nom, c'est un fichier,

  quand il y a *rep*, c'est un répertoire.

  <chemin>fich est un chemin qui désigne le fichier de nom fich.

  <chemin> peut être réduit à ε.

  Exemples :

  dans */Public/123 Public/TP1*/*fich1* :

  <chemin>=*/Public/123 Public/TP1/* et fich=*fich1*

  Dans *fichier1*, <chemin>=*ε* et fich=*fichier1*.

Manipulation de fichiers

1. **cp $fich_1$ $fich_2$**
2. **cp <$chemin_1$>$fich_1$ <$chemin_2$>$fich_2$**
3. **cp <$chemin_1$>fich <$chemin_2$>rep**
4. **cp <$chemin_1$>$fich_1$ … <$chemin_n$>$fich_n$ <$chemin_{dest}$>rep**
5. **cp <$chemin_1$>* <$chemin_{dest}$>rep**
6. **cp -r <$chemin_{source}$>rep_{source} <$chemin_{dest}$>rep_{dest}**

Manipulation de fichiers

- **Copie, déplacement et effacement de fichiers (suite)**

Renommage/déplacement de fichiers

mv (move) permet :

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

Syntaxe :

Changement de nom : mv *nom_fichier nouv_nom_fichier*

Exemple :

mv index.html accueil.html

Manipulation de fichiers

- **Copie, déplacement et effacement de fichiers (suite)**

Déplacement de fichiers

Syntaxe :

Déplacement : mv *source destination*

Différent si destination existe ou non, et si fichier ou répertoire

Exemple :

mv index.html /home/site/

Manipulation de fichiers

- **Copie, déplacement et effacement de fichiers (suite)**

Effacement ou suppression de fichiers ou répertoires

rm (remove) permet de supprimer des fichiers ou répertoires

Option –R ou -r pour supprimer un répertoire (et ce qu'il contient)

Attention : pas de corbeille …

Syntaxe :

rm *chemin*

Exemple :

rm /home/site/index.html

rm –R /home/site

Les métacaractères

- **Outils permettant d'optimiser les actions dans une commande**
  - **Les métacaractères sont symbolisés par les caractères suivants :**

    ***, ?, [ ], { }**

  - **Métacaractère ***
    - Remplace n'importe quelle suite de caractères (vide y compris)

    Exemple : lister tous les fichiers dont la 1ere lettre commence par a

    ls a*

  - **Métacaractère ?**
    - Remplace 1 et 1 seul caractère

    Exemple : lister tous les fichiers dont la dernière lettre est un et un seul caractère (tp1, tp2, tp3,… mais pas tp, tp12, tp1.txt …)

    ls tp?

Les métacaractères

- **Outils permettant d'optimiser les actions dans une commande**

  - **Métacaractère [ ]**

    Définition d'un intervalle de lettre ou chiffres.
    Remplace 1 seul caractère

    Exemples :

    lister tous les fichiers dont la première lettre commence par a, b ou c

    ls [abc]* ou ls [a-c]* (ou ls {a,b,c}*)

    lister tous les fichiers qui se termine par 7, 8 et 9

    ls *[7-9]

    lister tous les fichiers qui ne se termine pas par 6, 7 ,8, 9

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

Les métacaractères

- **Outils permettant d'optimiser les actions dans une commande**

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 :

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

Commandes/caractères spéciaux

- Ctrl-C (^C) **arrête un processus**
- Ctrl-D **exit (logout) (utilisation non conseillé)**

- Job & **exécute le processus en background (arrière-plan)**
- Ctrl-Z **suspend un processus**
- bg **bascule le processus en background (arrière-plan)**
- fg **permet de reprendre (en premier plan) l'exécution du job suspendu**

si on a oublié &, faire Ctrl-Z puis bg

- Ctrl-L **rafraichit l'écran (*clear* , *reset*)**

Système d’exploitation – Interpréteur de commandes

Patrick FULCONIS

Système d'exploitation

Environnement utilisé en TP lors de INF111 :

–un ensemble de terminaux-X ou de PC connectés à un réseau

–session de travail = ouvrir une connexion avec une des machines du réseau (un "serveur'", par exemple turing)

Le serveur contient donc un ensemble de programmes qui lui permettent :

–d'attendre des commandes, de les analyser et d'exécuter les programmes correspondants

–d'éditer des fichiers, de compiler des programmes C, etc.

–de gérer les ressources de la machine : disques, périphériques, processeur (partagés par différents utilisateurs)

–de gérer les utilisateurs (comptes, droits, ...)

–etc.

→ cet ensemble (fini !) de programmes constitue ce que l'on appelle le "**système d'exploitation**" (système **Unix** dans le cas des serveurs du DLST)

Rq : sur les PC du DLST, on se connecte sur le système d'exploitation Windows

SE - Interpréteur de commandes

Un des premiers rôles du système d'exploitation est d'exécuter en permanence un programme de dialogue avec l'utilisateur chargé d'attendre et d'exécuter des commandes.

Ces commandes peuvent être fournies soit à travers une interface graphique (par exemple en cliquant dans un menu), soit sous forme de texte saisi dans une fenêtre (soit même éventuellement sous d'autres formes : micro, caméra, etc).

→ l'important est de comprendre que, quelle que soit l'interface utilisée, ce dialogue est géré par un programme qui s'exécute en permanence sur la machine, **l'interpréteur de commandes**.

Dans le cas d'Unix, les commandes sont fournies (par défaut) à travers une interface textuelle. L'interpréteur de commandes s'appelle alors un "shell"

Le Shell

- Dans l'environnement du **shell**, chaque ligne démarre par un **prompt**,
- Celui-ci se termine par le caractère **#** si l'utilisateur connecté est root
  (administrateur)
- Par le caractère **$** s'il s'agit d'un autre utilisateur

Unix, Linux différencie les majuscules des minuscu

Unix n'aime pas les espaces **et** caractères spéciaux **dans les noms de fichiers, ni les** accents

L'espace sert de séparateur entre commandes/options/arguments

Différents shell : **bash**, sh, zsh, tcsh, …

Analyse et exécution d'une commande Unix

Le comportement par défaut de l'interpréteur de commande peut être décrit de la manière suivante :

1. attente d'une séquence de caractères terminée par "Return"

2. analyse de cette séquence : est-ce une commande Unix correcte ?
— si oui, on l'exécute, et on attend que cette exécution se termine !
— sinon, on affiche un message d'erreur

3. on recommence ...

Analyse et exécution d'une commande Unix

Dans certains cas, on souhaite ne pas attendre que la commande en cours d'exécution soit terminée avant de pouvoir exécuter une nouvelle commande :

– soit parce que c'est une commande qui prend du temps (un calcul long),

– soit parce que l'on veut que cette commande continue à s'exécuter (nedit).

On peut alors indiquer à l'interpréteur que cette commande doit être exécutée en "tâche de fond" (*background*) en la faisant suivre de '**&**'. 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 **&**

variante possible (si on a oublie le &) :

nedit (le shell est « bloqué »)

CTRL-Z (suspend l'exécution de la commande en cours)

bg (relance en tâche de fond (background) la dernière commande suspendue)

(en général : le shell affiche « gedit & »)

Analyse et exécution d'une commande Unix

Remarque :

On peut ainsi avoir plusieurs commandes qui s'exécutent en même temps : il faut donc que la machine soit capable d'exécuter plusieurs programmes "en même temps" alors qu'elle ne possède en général qu'un petit nombre de processeurs (voire un seul !).

En pratique cette "simultanéité" est obtenue à travers une fonction du système d'exploitation 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.

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

Analyse et exécution d'une commande Unix

Pour faciliter l'analyse, les commandes Unix sont toujours de la même forme : un <nom de commande> suivi de 0 ou plusieurs <arguments>

exemples :

cp tp1.c tp2.c

mkdir TP

nedit un_fichier

gcc tp1.c

ls

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

— lorsque le <nom de commande> est incorrect

— lorsque les <arguments> sont incorrects (pas le bon nombre, "options" inexistantes)

— lorsque la commande ne peut s'exécuter correctement (les arguments sont "incorrects" : fichiers manquants, problème de droits, etc.)

Analyse et exécution d'une commande Unix

- **En particulier, quel genre d'erreur obtient-on si 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)**

Analyse et exécution d'une commande Unix

- **Remarque 1 :**
  - **"exécuter" une commande signifie en géneral démarrer un (nouveau) programme chargé d'exécuter cette commande. (c'est le cas pour ls, mkdir, etc.).**
  - **Certaines autres commandes sont traitées directement par l'interpréteur de commande (cd). Mais tout ça est assez transparent pour l'utilisateur ...**
  - **Une question connexe est de savoir comment fait l'interpréteur de commande pour savoir "où se trouve" le programme à exécuter ? la réponse dans un prochain épisode !**

Analyse et exécution d'une commande Unix

- **Remarque 2 :**
  - **Il y a des commandes qui "qui modifie quelque chose" dans l'environnement (comme cd ou mkdir ou cp) et**
  - **Des commandes qui "ne modifie rien" mis à part les pixels de l'écran. (comme ls ou pwd).**

Linux

Droits et utilisateurs

Patrick FULCONIS

Utilisateurs et permissions

- **Unix distingue 3 catégories d'utilisateurs**
  - User (u)
  - Group (g)
  - Others (o)

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

- Others (o)
  - Désigne tous les autres utilisateurs

Utilisateurs et permissions (suite)

- **Unix distingue 3 types de permissions**
  - **r** : read => permission de lecture
  - **w** : write => permission d'écriture
  - **x** : exécution => permission d'exécution

- **Visualiser les permissions sur les fichiers/répertoires**

La commande pour visualiser les permissions est : *ls -l*

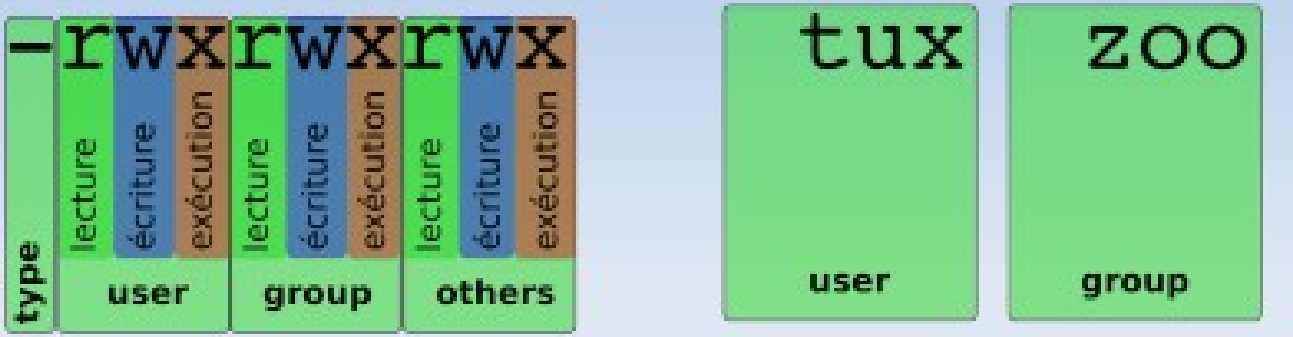
Type	Propriétaire U	Groupe G	Autres O	Nom propriétaire	Groupe	Nom fichier
d	r w x	r - x	r - x	ubuntu	prof	cours
-	r w x	r w x	r w x	ubuntu	prof	img.jpg
-	r w x	- - -	- - -	ubuntu	prof	tp_unix.txt

Utilisateurs et permissions (suite)

- **Types de fichiers**
  - **-** : fichier normal
  - **d** : répertoire
  - **b** : fichier spécial – mode blocs (disquette, clé usb, ...)
  - **c** : fichier spécial – mode caractères (ram, imprimante, ...)
  - **l** : lien symbolique (raccourci)
  - **s** : socket (communications entre processus ou réseau)
  - **p** : pipe (communications entre programmes)

Utilisateurs et permissions (suite)

```
user@host:~$ ls -la /fichier
```



Type		Droit		Destinataire	
b	Block device	r	Read	u	*User (propriétaire)*
c	Character device file	w	Write	g	*Groupe*
d	Répertoire (Directory)	x	Execute (fichier)	o	*Others (les autres, ni 'u' ni 'g')*
l	Lien symbolique	x	chdir (répertoire)		
s	Socket	s	SUID bit		
p	FIFO				
-	Fichier normal				

Utilisateurs et permissions (suite)

- **Sémantique des permissions**

Fichier	r	Autorise la consultation – Affichage – copie
	w	Autorise la modification – modification du contenu – suppression du fichier – changement de nom
	x	Autorise l'exécution – le fichier doit être un programme

Utilisateurs et permissions (suite)

- **Sémantique des permissions**

Répertoire	**r**	**Autorise la consultation** – affichage du contenu (*ls*)
	w	**Autorise la modification** – modification du nom (*mv*) – ajout/suppression du contenu (*rm*) – suppression du répertoire (*rm*)
	x	**Autorise la traversée** – utilisation dans un chemin d'accès à une entrée (fichier/répertoire) (*cd*)

Utilisateurs et permissions (suite)

- **Modification des droits**
  - **La commande *chmod* (change mode) permet de modifier et d'attribuer des droits.**

- **Syntaxe**

  chmod qui(u g o) quoi(+ - =) permission(r w x) **fichier/rep**

  a = all = ugo (chmod +**r** = chmod a+**r** = chmod ugo+**r**)

  Exemple 1 : retirer les droits d'écriture et d'exécution pour le groupe et les autres sur le fichier img.jpg

  chmod go-wx img.jpg

  Exemple 2 : ajouter les droits de lecture et d'écriture pour le groupe sur le fichier tp_unix.txt

  chmod g+rw tp_unix.txt

Utilisateurs et permissions (suite)

- **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

Ex: chmod 764 toto.sh

FIN

- Merci de votre attention !