Annexe A: Unix/Linux

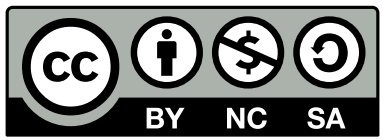
INF3135

Construction et maintenance de logiciels

Alexandre Blondin Massé

Université du Québec à Montréal

v253



Notes préalables

- Rappels **condensés** des 3 premiers chapitres du cours INF1070 Utilisation et administration des systèmes informatiques
- Le matériel **original** a été créé par **Jean Privat** et **Alexandre Blondin Massé**
- Certaines **modifications** ont été apportées

Plan

Introduction

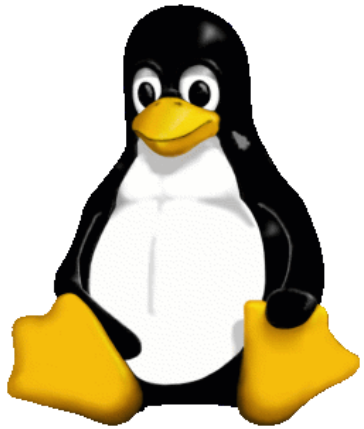
Début d'UNIX

- 1969 Ken Thompson et Dennis Ritchie développent un Unix primitif (chez Bell)
- 1972 Ritchie invente le C & Thompson réécrit Unix en C

GNU (GNU is not UNIX)

- 1984 Richard Stallman annonce le projet GNU.
  Développement de logiciels Unix libres:
  compilateur C (gcc) et autre outils
- 1985 Stallman crée la *Free Software Foundation* (FSF)
- 1989 Stallman publie la *General Public License* (GPLv1)

Linux



- 1991 Linus Torvalds annonce le développement d'un noyau UNIX libre pour PC Intel 80386
- Le projet GNU réimplémentait tous les utilitaires, mais il lui manquait un noyau

Distribution Linux

- Système d'exploitation **complet**
- Ensemble **cohérent** de logiciels
- Basé sur un noyau **linux** et des outils **GNU**
- Organisation et processus de **publication**
- **Outils** d'installation et mise-à-jour
  dont le **gestionnaire de paquets**

Plus de 300 distributions actives existent (selon distrowatch)

- Linux Lite, Zorin OS
- Elementary OS, Trenta OS, Deepin
- Qubes OS, Tail OS

Gestionnaire de paquets

- Logiciel gérant l'**installation** et la **mise-à-jour** de logiciels
- Centralise et **simplifie** grandement la gestion
- Gère les **dépendances** entre paquets
- Maintient l'**historique** des installations et des mises-à-jour

Exemples

- Linux : `apt` (Debian et dérivés), `pacman` (Arch Linux), `dnf`/`yum` (Redhat et cie.)...
- MacOS : MacPorts, Homebrew
- Windows : winget (récent, peu utilisé encore), Chocolatey, NuGet

Dans le cours: `apt`

Shell et terminal

Shell et terminal

Shell: le programme qui **interprète les commandes**

Le shell lit des commandes et les exécute

- `bash` (*Bourne-Again shell*) de GNU (1989) — le plus commun
- `ash` (*Almquist shell*) (1989) — minimaliste (embarqué, scripts)
- `dash` (*Debian Almquist shell*) (2002) — conforme à POSIX et dérivé de ash
- `PowerShell` de Microsoft (2006)

Terminal: l'**interface** physique (ou virtuelle)

Un terminal c'est un clavier et un écran texte

- Fenêtre de terminal (émulateur de terminal)
  Exemples: `xterm`, `gnome-terminal`, `konsole`, `iTerm2`, `cmder`
- Autre **pseudo-terminal**: `ssh`, `screen`, `tmux`
- Terminal **physique**

Par défaut, l'émulateur de terminal exécute un shell

Fonctionnement du shell

Le shell est un programme comme les autres

1. Affiche l'invite de commande
2. Lit la commande de l'utilisateur
3. Analyse la ligne de la commande et ses caractères spéciaux
4. Exécute la commande
5. Recommence au point 1

Quoi faire avec le shell?

- Naviguer dans et interagir avec le système de fichiers
- Exécuter et contrôler des commandes et des utilitaires
- Développer des petits programmes (scripts shell)

Invite de commande shell

Invite (*prompt*): indique que l'utilisateur peut entrer des commandes

```
privat@lama:~/ens/INF1070$
```

- Nom de l'utilisateur: `privat`
- Nom de la machine: `lama`
- Répertoire courant: `~/ens/INF1070`
- Indicateur d'utilisateur (à la fin)
  - « $ » (utilisateur normal)
  - « # » (super-utilisateur)
- Invite secondaire: « > » quand le shell a besoin de plus de saisies

L'invite est configurable

Quelques commandes de base

- `echo` — afficher un message
- `ls` — lister le contenu du répertoire
- `cat` — afficher un fichier
- `cd` — changer de répertoire
- `pwd` — afficher le nom du répertoire courant
- `exit` — fermer le shell

```
$ ls
hello.txt lisez-moi.txt  répertoire
$ cat lisez-moi.txt
Bash est un interpréteur [...]
$ cd répertoire
$ ls
document.txt
$ cat document.txt
INF1070 - Utilisation et [...]
$ exit
```

Autres commandes d'affichage

- `head` — afficher les première lignes
- `tail` — afficher les dernières lignes
- `less` (et `more`) — afficher un fichier page par page
- `nl` — numéroter les lignes
- `tac` — afficher un fichier en commençant par la dernière ligne
- `rev` — inverser chacune des lignes affichées
- `wc` — compter le nombre de lignes, mots et octets

```
$ wc fichier1 fichier2
$ wc --lines fichier1 fichier2
$ wc -l fichier1 fichier2
$ wc --words fichier1 fichier2
$ wc -w fichier1 fichier2

$ head --lines 2 fichier1 fichier2
$ head --lines=2 fichier1 fichier2
$ head -n 2 fichier1 fichier2
$ head -n2 fichier1 fichier2
```

Avoir de l'aide sur une commande

- Le `man`
- Par défaut, utilise le paginateur (*pager*) `less` pour naviguer dans le document

Aide-mémoire:

- [q] quitte
- [Espace] ou [Page↓] défile d'une page
- [⏎] ou [↓] défile d'une ligne
- [g] ou [Home] va au début
- [G] ou [End] va à la fin
- un nombre et [⏎] avance du nombre de lignes indiquées
- `/motif` cherche l'occurrence suivante de `motif`
- [n] cherche l'occurrence suivante
- [N] cherche l'occurrence précédente
- [h] pour un résumé des commandes et des raccourcis clavier

Éditeur de texte

- Logiciel facilitant la manipulation de fichiers textes
- Texte brut (sans mise en forme) $\neq$ traitement de texte
- Police à chasse fixe pour l'alignement vertical (indentation)
- Utilisé pour la programmation (code source)
- Utilisé pour l'administration système (fichiers de configuration)

L'offre d'éditeurs de texte est très variée

- Notepad/Notepad++ (Windows)
- TextEdit (MacOS)
- gedit (multiplateforme)
- SublimeText (multiplateforme)
- Visual Studio Code (multiplateforme)
- Emacs et ses dérivés (multiplateforme)
- Vi/Vim et ses dérivés (multiplateforme)
- nano (Unix)
- ed le premier éditeur texte Unix ($\approx$ 1969), toujours disponible

Vi/Vim/Neovim

L'éditeur utilisé en classe

Caractéristiques:

- Un des plus anciens éditeurs de texte
- Un des éditeurs de texte les plus utilisés dans le monde
- Ancêtre: Vi, créé par Bill Joy en 1976
- Vim = Vi iMproved
- Multiplateforme (Linux, MacOS, Windows)
- Standard sous UNIX ($\approx$ déjà installé)

Arguments des commandes

Chaque commande traite ses arguments

- De sa façon **spécifique**
- Lisez le manuel

Il y a quand même des conventions

- Des options: qui commencent par « - »
- Le reste des arguments: qui ne commencent pas par « - »

Attention

Chaque commande peut avoir une gestion spécifique des arguments

Options des commandes

- Activent certains comportements spécifiques
- Configurent certains paramètres
- Sont combinables

Options courtes

- Commencent par « - » (tiret)
- Une lettre (ex. « `cat -n` »)
- S'agglutinent (ex. « `cat -nE` » vaut « `cat -n -E` »)
- Note: `-n` et `-E` sont des extensions GNU

Options longues (style GNU)

- Commencent par « -- » (deux tirets)
- Ont parfois un synonyme court
- Exemple: « `cat --number --show-ends` »

Conventions

Certaines options sont comprises par de nombreuses commandes

- `--help` — affiche l'aide
- `--verbose` — mode verbeux
- `--version` — affiche la version de la commande

Rappel: chaque commande a ses propres règles

Complètement (*completion*)

Le *shell* `bash` peut aider à écrire les commandes

Tabulation simple ⟶| (ou tab)

- complète l'argument ou l'option (si possible)
- `cat /etc/pas` ⟶|
→ complète: `cat /etc/passwd`

Tabulation double ⟶| ⟶| (ou tab-tab)

- affiche une liste d'options ou d'arguments possibles
- `ls --re` ⟶| ⟶|
→ propose: `--recursive --reverse`

Bash adapte le complètement aux commandes qu'il connaît
Certains shells sont plus avancés (`zsh`, `powershell`)

Redirection en sortie

Le résultat des commandes va sur la **sortie standard**

- Par défaut, la sortie standard est l'écran terminal
- Mais le **shell** peut la rediriger vers un fichier

```
$ ls -l hello.txt > ls.out
$ cat ls.out
-rw-r--r-- 1 privat privat 460 aoû 15 20:23 hello.txt
```

Redirection en sortie

Opérateurs shell de redirections: « > » et « >> »

- « `> f` » écrit dans le fichier `f` depuis le début (écrase, *overwrite*)
- « `>> f` » écrit à la suite du fichier `f` (ajoute, *append*)
- Dans les deux cas, si `f` n'existe pas, il est créé

```
$ echo uno > tata
$ echo dos >> tata
$ cat tata
uno
dos
```

Sortie d'erreur standard

Les commandes distinguent deux sorties

- la **sortie standard** pour les résultats normaux
- la **sortie d'erreur standard** pour les messages d'erreurs et de diagnostics

```
$ ls -l hello.txt epic.fail > ls.out
ls: impossible d'accéder à 'epic.fail':
Aucun fichier ou dossier de ce type
$ cat ls.out
-rw-r--r-- 1 privat privat 460 aoû 15 20:23 hello.txt
```

Redirection d'erreur standard

« `2>` » (et « `2>>` ») redirigent la **sortie d'erreur standard** vers un fichier

```
$ ls -l hello.txt epic.fail > ls.out 2> ls.err
$ cat ls.out
-rw-r--r-- 1 privat privat 460 aoû 15 20:23 hello.txt
$ cat ls.err
ls: impossible d'accéder à 'epic.fail':
Aucun fichier ou dossier de ce type
```

/dev/null

- Un fichier spécial qui accepte (et ignore) des données
- On l'utilise pour ignorer des sorties de commande
- Souvent utilisé pour *taire* la sortie d'erreur standard

```
$ ls -l hello.txt epic.fail 2> /dev/null
-rw-r--r-- 1 privat privat 460 aoû 15 20:23 hello.txt
```

Tubes

Le « | » (tube, *pipe*) connecte des commandes

- La **sortie standard** de l'une est connectée à
- l'**entrée standard** de la suivante

```
$ echo bonjour le monde | wc
1 3 17
$ echo bonjour le monde | rev | lolcat
ednom el ruojnob
```

Une **conduite** (*pipeline*) est une suite de commandes connectées par des tubes

Tubes et filtres

De nombreuses commandes traitent, filtrent ou transforment l'entrée standard vers la sortie standard

- `head`, `tail`, `less`, `tac`, `rev`, `wc`
- `sort` – trier les lignes
- `uniq` – éliminer les lignes répétées
- `tr` – convertir ou éliminer des caractères
- `grep` – afficher les lignes correspondant à un motif
- `cowsay` – vache qui parle (extra)
- `lolcat` – colorer en arc-en-ciel (extra)

Fichiers en arguments et entrée standard

La plupart des commandes traitent les fichiers et l'entrée standard

Convention habituelle des commandes

- Si plusieurs fichiers
  $\rightarrow$ Ils sont traités dans l'ordre
- Si un fichier n'existe pas
  $\rightarrow$ La commande affiche un message d'erreur
- Si pas de fichier
  $\rightarrow$ La commande lit l'*entrée standard*
- Si un argument est « `-` » (tiret seul)
  $\rightarrow$ Ça désigne aussi l'entrée standard

Convention $\neq$ standard

- Chaque commande a son propre comportement
- Lisez le manuel (`cat` par exemple)

L'entrée standard au clavier

L'**entrée standard** est l'entrée naturelle des commandes

- Par défaut, l'entrée standard est le clavier du terminal
- Les touches ctrl + D terminent l'entrée clavier
  Prononcer « contrôle-dé », écrire « `^D` »

```
$ tac
bonjour
le monde
^D
le monde
bonjour
```

Commandes interactives

- Les commandes **interactives** dialoguent avec l'utilisateur
- Il saisit des instructions ou répond aux questions via le terminal

`sh` — démarrer un interpréteur de commande (*shell*)

```
$ sh
$ echo hello
hello
$ exit
```

`python` — lancer l'interpréteur du langage de programmation Python

```
$ python3
>>> print(1+1)
2
>>> quit()
```

Redirection en entrée

« < » redirige l'**entrée standard** depuis un fichier

```
$ cat < hello.txt
Bonjour [...]
```

Nécessaire pour lire un fichier avec certaines commandes

Développement des noms de fichiers (glob)

Désigner simplement un ensemble de fichiers selon un motif

- Point d'interrogation `?` — un caractère quelconque
- Étoile `*` — zéro, un ou plusieurs caractères
- Crochets `[]` — un seul des caractères de la liste

Exemples

- « `cat *.txt` » se terminent par `.txt`
- « `cat ?[oa]*` » la deuxième lettre est `a` ou `o`

Note

- Le glob est géré par le shell (pas par la commande)
- La commande *ne voit* que les arguments une fois développés

Les détails: le manuel de `glob(7)`

Caractères spéciaux

Un **caractère spécial** est un caractère qui n'a pas un sens littéral

Exemples

- « `^D` » et « `^C` » sont spéciaux pour le terminal
- « `*` » et l'espace sont spéciaux pour le shell
- « `-` » en début d'argument est spécial pour la plupart des commandes

Difficultés

- Le sens d'un caractère dépend du contexte
- Chaque commande a ses règles
- Les règles sont + ou - compliquées
- Forcer l'interprétation littérale est + ou - complexe

Échapper avec la contre-oblique

- Échapper avec « \ » (contre-oblique, *backslash*)

→ Annule le caractère spécial qui suit

```
$ echo \*
*
```

Le « \ » s'utilise pour échapper des caractères dans d'autres contextes

Encadrer avec les guillemets simples

- « ' » (guillemet simple, *simple quote*)

→ Force l'interprétation littérale jusqu'au prochain « ' »

```
$ echo '*   \'
*   \
```

On peut utiliser « \ » sur le premier « ' » pour l'ignorer

```
$ echo \'
'
```

Encadrer avec les guillemets doubles

- « " » (guillemets doubles, *double quote*)

→ Une version amoindrie de « ' »

→ Permet 3 caractères spéciaux internes: « \ », « $ » et « ` » (on les verra plus tard)

```
$ echo "*    \"    \\"
*    "    \
```

Attention! Ne pas confondre avec les chaînes de caractères des langages de programmation

Caractères spéciaux des commandes

- Chaque commande a ses propres règles
- Et ses propres options qui changent les règles

```
$ echo 'Une ligne\n\tbrisée'
Une ligne\n\tbrisée
$ echo -e 'Une ligne\n\tbrisée'
Une ligne
        brisée
```

Note: `echo -e` n'est pas POSIX, `printf` l'est.

Caractères spéciaux des commandes

Par convention « `--` » désigne la fin des options

```
$ cat -n world.txt
     1  Bonjour
     2  Monde
$ cat -- -n
Un bon numéro
$ cat -- -n world.txt
Un bon numéro
Bonjour
Monde
```

Fichiers

Fichier = information numérique = donnée

Manipulation des fichiers via des applications

- Ouverture via l'application (menu Fichier > Ouvrir)
- Exécution d'une commande avec le fichier en argument

```
$ vim readme.txt
$ display debian.png
$ evince man.pdf
```

Application par défaut (par type de fichier)

- Double-clic sur l'icône du fichier lance l'application par défaut
- `xdg-open` ouvre un fichier dans l'application par défaut

```
$ xdg-open debian.png
```

Lister les fichiers

Commande `ls` (*list*)

- Affiche le contenu des répertoires passés en argument
- Ou le répertoire courant par défaut

```
$ ls
hello.txt  lisez-moi.txt  répertoire
```

- `-R` (`--recursive`): parcours les sous-répertoires
- `-d` (`--directory`): affiche les répertoires (et non leur contenu)

`tree` (extra) — affiche l'arborescence

```
$ tree --charset=ascii
data/base/
|-- hello.txt
|-- lisez-moi.txt
|-- README.md
`-- répertoire
    `-- document.txt
```

Répertoire personnel (ou maison)

Chaque utilisateur a un répertoire de connexion (*home directory*)

- `cd` sans argument y mène

Caractère spécial « ~ » (tilde) du *shell*

- `~` seul devient le répertoire de connexion
- `~/` en début d'un argument fonctionne aussi
- `~toto` seul devient le répertoire de connexion de l'utilisateur `toto`
- `~toto/` en début d'un argument fonctionne aussi

Répertoire courant

Commande `pwd` (*print working directory*)

- Afficher le chemin complet du répertoire courant

Commande interne `cd` (*change directory*)

- Change le répertoire de travail courant
- « `cd toto` » Va au sous-répertoire `toto`
- « `cd ..` » Remonte au répertoire parent
- « `cd` » Retourne au répertoire de connexion
- « `cd -` » Retourne au répertoire précédent (et l'affiche)

Attention à l'espace entre `cd` et `..` (ou `cd` et `-`)

Déplacer/renommer des fichiers

`mv` (*move*) déplace/renomme des fichiers (ou répertoires)

- `mv ancien nouveau`
- `mv f1 f2 f3 répertoire`

Attention: « `mv foo bar` » peut être ambigu

- Si `bar` n'existe pas: `foo` est **renommé** en `bar`
- Si `bar` est un fichier régulier: `bar` est **écrasé** par `foo`
- Si `bar` est un répertoire: `foo` est **déplacé** dans `bar`
  c'est équivalent à « `mv foo bar/foo` »

Autres commandes de manipulation

`cp` (*copy*) copier (ou écraser) des fichiers

- `-R` (`-r`, `--recursive`) copier récursivement des répertoires
- `-p` (`--preserve`) préserver des attributs (voir plus loin)
- `-a` (`--archive`) récursif et préserver tous les attributs (GNU)
- `-i` ou `-f`: écrasement interactif ou forcé

`rm` (*remove*) supprimer des fichiers (pas de corbeille)

- `-R` (`-r`, `--recursive`) supprimer récursivement des répertoires
- `-i` ou `-f`: suppression interactive ou forcée

Commandes de manipulation des répertoires

`mkdir` (*make directory*) crée des répertoires (vides)

- `-p` (`--parents`) créer les répertoires parents intermédiaires

`rmdir` (*remove directory*) supprime des répertoires vides

- `-p` (`--parents`) supprimer les répertoires parents vides intermédiaires

Texte et binaire

Fichier **texte**

- Contenu = une séquence de caractères (imprimables)
- Peut se lire et se modifier avec éditeur de texte
- Codages: ASCII, ISO-8859-15, UTF-8, etc.
  man `ascii`
- Représentations des fins de lignes: Unix (`\n`) et DOS (`\r\n`)

Fichier **binaire**

- Fichier non-texte
- Parfois difficile de trancher entre texte/binaire
- Mais généralement non ambigu

Comparaison de fichiers

- `cmp` – Compare deux fichiers octet par octet
- `diff` – Compare deux fichiers ligne par ligne

```
$ cmp /bin/bash /bin/dash
/bin/bash /bin/dash differ: byte 26, line 1

$ cmp /bin/dash /bin/sh

$ diff /bin/bash /bin/dash
Binary files /bin/bash and /bin/dash differ
```

Fichiers sous Unix

Un principe Unix de base: tout est fichier

Fichiers réguliers

- Textes, exécutables, code source, images...
- Contenu décidé par l'utilisateur

Fichiers spéciaux

- Répertoires, fichiers périphériques (dans `/dev`), liens symboliques, tubes nommés...
- Manipulation par des commandes spéciales
- Règles au cas par cas

Méta-données des fichiers (attributs)

Autres options de `ls`:

- `-l` format d'affichage long
- `-i` (`--inode`) afficher le numéro d'index

```
$ ls -li /etc/passwd
30169 -rw-r--r-- 1 root root 2652 nov 16 2017 /etc/passwd
```

- « `30169` » numéro d'index (inœud, *inode*)
- « `-` » type de fichier (fichier régulier, répertoire…)
- « `rw-r--r--` » droits (utilisateur, groupe, autre)
- « `1` » nombre de liens durs
- « `root` » utilisateur propriétaire
- « `root` » groupe propriétaire
- « `2652` » taille du fichier (en octets)
- « `nov 16 2017` » date de dernière modification

Trier les fichiers

Autres options de `ls`:

- `-S` trier par taille
- `-t` trier par date de modification
- `-r` (`--reverse`) trier à l'envers

```
$ ls -lSr
-rw-r--r-- 1          72 sep 12 20:49 hello.c
-rw-r--r-- 1         460 sep 12 20:49 bonjour.txt
-rw-r--r-- 1        6863 sep 12 20:49 debian.svg
-rw-r--r-- 1       15197 sep 11 18:50 debian.png
-rw-r--r-- 1       41493 sep  6 20:30 linux.au
-rw-r--r-- 1 2729345024 sep 10 16:01 debian.iso
```

État complet d'un fichier

`stat` — affiche l'état complet d'un fichier (GNU)

- `-c` (`--format`) utilise le format indiqué

```
$ stat /etc/passwd
Fichier : /etc/passwd
Taille : 2652  Blocs : 8  Blocs d'E/S : 4096  fichier
Périphérique : 807h/2055d  Inoeud : 30169  Liens : 1
Accès : (0644/-rw-r--r--)  UID : (0/root)  GID : (0/root)
Accès  : 2018-06-05 09:39:01.288330072 -0400
Modif. : 2017-11-16 09:58:36.656200344 -0500
Changt : 2017-11-16 09:58:36.680200043 -0500
Créé   : -

$ stat -c '%n %U %s' /etc/passwd
/etc/passwd root 2652
```

Taille des fichiers

Unité de base: l'**octet** (**byte**), qui contient 8 **bits**

- 1 octet = 1o
- 1 byte = 1B
- 8 bits = 8b

→ Attention à la **confusion** byte et bit

Lister la taille des fichiers

Autres options de `ls` (GNU):

- `-h` (`--human-readable`) afficher les tailles en multiples de 1024
- `--si` afficher les tailles en multiples de 1000

```
$ ls -lh
-rw-r--r-- 1  460 sep 12 13:31 bonjour.txt
-rw-r--r-- 1 2,6G sep 10 16:01 debian.iso
-rw-r--r-- 1  15K sep  7 11:36 debian.png
-rw-r--r-- 1 6,8K jun  1 00:50 debian.svg
-rw-r--r-- 1  41K fév 15  2000 linux.au

$ ls -l --si
-rw-r--r-- 1  460 sep 12 13:31 bonjour.txt
-rw-r--r-- 2 2,8G sep 10 16:01 debian.iso
-rw-r--r-- 1  16k sep  7 11:36 debian.png
-rw-r--r-- 1 6,9k jun  1 00:50 debian.svg
-rw-r--r-- 1  42k fév 15  2000 linux.au
```

Droits et utilisateurs

Les fichiers et répertoires ont des **propriétaires** et des **droits** qui limitent/autorisent l'accès à certains **utilisateurs**.

- Chaque utilisateur est associé à une identité et à des groupes
- Chaque fichier du système possède un utilisateur propriétaire et un groupe propriétaire

Par défaut et en général:

- Un utilisateur peut lire les fichiers des autres utilisateurs
- Un utilisateur ne peut modifier les fichiers des autres utilisateurs
- Un utilisateur peut créer des fichiers dans son répertoire personnel

Super-utilisateur

Le super utilisateur (*super-user*)

- S'appelle **root** (traditionnellement)
- Est le compte de l'administrateur
- Son uid est 0

Il a de grands pouvoirs: contrôle **complet** du système

- Ne respecte pas les droits des fichiers
- Peut contrôler les programmes des utilisateurs
- Peut configurer le système et le réseau
- Peut installer de nouveaux programmes et pilotes

Et de grandes responsabilités

- L'administrateur a aussi un compte utilisateur normal
- Ne passe en **root** que lorsque c'est nécessaire

Passer en root

`sudo` (extra) – exécute une commande sous un autre utilisateur

- Moderne (et le défaut de nos jours)
- Change temporairement d'utilisateur (root par défaut)
- Exécute une commande (défaut)
- Nécessite une configuration préalable
- Nécessite le mot de passe de l'**utilisateur courant**
- `-s` pour lancer un shell
- Journalise (*log*) les utilisations

```
$ sudo apt install cowsay
[...]
$ grep sudo /var/log/auth.log
Sep 12 22:38:43 lama sudo:   privat
  PWD=/home/privat/ens/INF1070/slides
  COMMAND=/usr/bin/apt install cowsay
```

Droits des fichiers traditionnels Unix

3 catégories d'accès (ugo)

- u (*user*/utilisateur) l'utilisateur propriétaire
- g (*group*/groupe) le groupe propriétaire
- o (*other*/autre) les autres

3 permissions par catégorie (rwx)

Pour un fichier

- r (*read*/lecture) on peut lire le contenu
- w (*write*/écriture) on peut modifier le contenu
- x (*execute*/exécution) on peut exécuter le programme

Pour un répertoire

- r (*read*/lecture) on peut lister les entrées
- w (*write*/écriture) on peut ajouter/supprimer des entrées
- x (*execute*/exécution) on peut accéder aux entrées

Modifier les droits

Commande `chmod` (*change file mode*)

- `chmod octal fichier`
- `chmod mode[,...] fichier`

où `mode` contient

- les catégories d'utilisateurs: **u**, **g**, **o** ou **a** (*all*)
- une opération: **+** (ajouter), **-** (enlever), **=** (mettre exactement)
- les droits: **r**, **w**, **x**
- combinable avec **,**

Options utiles

- `-R` récursif
- `-v` verbeux

Exemple

```
$ ls -l fichier
-rw-r--r-- 1 privat privat fichier
$ chmod 640 fichier ; ls -l fichier
-rw-r----- 1 privat privat fichier
$ chmod +x fichier ; ls -l fichier
-rwxr-x--x 1 privat privat fichier
$ chmod o-x,g+w fichier ; ls -l fichier
-rwxrwx--- 1 privat privat fichier
$ chmod a-x fichier ; ls -l fichier
-rw-rw---- 1 privat privat fichier
```

Dates des fichiers (Unix)

Trois types de dates

- atime : date de dernier accès au fichier (lecture)
- mtime : date de dernière modification du fichier
- ctime : date de dernière modification des métadonnées

```
$ stat /etc/passwd
  Fichier : /etc/passwd
   Taille : 2720   Blocs : 8  Blocs d'E/S : 4096
fichier
Périphérique : 807h/2055d  Inoeud : 3020392  Liens : 1
Accès : (0644/-rw-r--r--)  UID : (0/root)   GID : (0/root)
 Accès : 2018-07-08 14:29:13.536125040 -0400
Modif. : 2018-06-20 09:20:56.939040056 -0400
Changt : 2018-06-20 09:20:56.963040057 -0400
  Créé : -
```

Dates (Unix)

Représentation: temps Unix

- Temps écoulé depuis *epoch*, le premier janvier 1970 UTC.
- Stocké en nanosecondes (historiquement en secondes)

```
$ stat -c '%.Y = %y' /etc/passwd
1529500856,939040056 = 2018-06-20 09:20:56.939040056 -0400

$ date +%s.%N
1536867622.139983577
```

Liens symboliques

Nouveau type de fichier « `l` » (L)

- Contenu : un chemin (relatif ou absolu)
- Même vers un fichier spécial : répertoire, fichier périphérique, un autre lien symbolique
- Même vers un autre système de fichiers

```
$ ls -lh /bin/sh
lrwxrwxrwx 1 root root    4 jan 24 2017 sh -> dash
$ ls -lh /bin/dash
-rwxr-xr-x 1 root root 115K jan 24 2017 dash
```

`readlink` — affiche la valeur d'un lien symbolique (GNU)

```
$ readlink /bin/sh
dash
```

Utilisation des liens symboliques

- **Ouvrir** un lien symbolique c'est ouvrir le fichier lié
  ouvrir = lire, écrire ou exécuter
- Le système d'exploitation suit les liens **automatiquement**
- Les programmes n'ont pas à les gérer **spécifiquement**
  (mais peuvent s'ils le veulent)

```
$ ls -l
lrwxrwxrwx 1 privat privat   16 lien.txt->les_effarés.txt
-rw-r--r-- 1 privat privat 1155 les_effarés.txt
$ wc lien.txt les_effarés.txt
  51  191 1155 lien.txt
  51  191 1155 les_effarés.txt
 102  382 2310 total
$ echo toto >> lien.txt
$ tail -n 2 les_effarés.txt
Au vent d'hiver.
toto
```

Création de liens symboliques

Commande `ln -s` (*link –symbolic*)

```
$ ln -s fichier lien
$ ls -il fichier lien
188577 -rw-rw-r-- 1 privat 0 jui  9 08:39 fichier
188786 lrwxrwxrwx 1 privat 7 jui  9 09:31 lien -> fichier
$ echo 'Bonjour' > lien
$ cat fichier
Bonjour
```

Manipulation de liens symboliques

Un lien symbolique est un fichier autonome

- Numéro d'index (inœud ou *inode*)
- Dates
- Propriétaires
- Pas de droits (toujours 0777)
- Indépendant du fichier lié éventuel

On a donc 2 fichiers

- le fichier lié
- le lien lui-même

Cibles des opérations : lien ou lié ?

Supprimer (`rm`), déplacer/renommer (`mv`)

- Le lien

Copier (`cp`)

- Le lié, par défaut
- `-P` (`--no-dereference`) pour copier le lien

Modifier (`touch`, `chown`, `chgrp`)

- Le lié, par défaut
- `-h` pour ne pas suivre le lien

Ouvrir, lire, écrire, exécuter

- Le lié, toujours