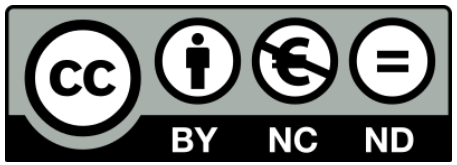
Unix/Linux utilisateur

Jean-Pierre Messager (jp@xiasma.fr)

18 novembre 2022 – version 1.1



Utilisation commerciale interdite sans autorisation

Conditions de distribution

Licence Creative Commons
Attribution - Pas d'Utilisation Commerciale
Pas de Modification 3.0 France
(CC BY-NC-ND 3.0 FR)

Ceci est un résumé de la licence complète disponible à :

https://creativecommons.org/licenses/by-nc-nd/3.0/fr/legalcode

Vous êtes autorisé à :

Partager — copier, distribuer et communiquer le matériel par tous moyens et sous tous formats.

L'offrant ne peut retirer les autorisations concédées par la licence tant que vous appliquez les termes de cette licence.

Selon les conditions suivantes :

Attribution — Vous devez créditer l'œuvre, intégrer un lien vers la licence et indiquer si des modifications ont été effectuées à l'œuvre. Vous devez indiquer ces informations par tous les moyens raisonnables, sans toutefois suggérer que l'offrant vous soutient ou soutient la façon dont vous avez utilisé son œuvre.

Pas d'utilisation commerciale — Vous n'êtes pas autorisé à faire un usage commercial de cette œuvre, tout ou partie du matériel la composant. Le titulaire des droits peut autoriser tous les types d'utilisation ou au contraire restreindre aux utilisations non commerciales *(les utilisations commerciales restant soumises à son autorisation.)*

Pas de modifications — Dans le cas où vous reprenez ce document dans une autre œuvre, que vous transformez, ou créez à partir du matériel composant l'œuvre originale, vous n'êtes pas autorisé à distribuer ou mettre à disposition l'œuvre modifiée.

Pas de restrictions complémentaires — Vous n'êtes pas autorisé à appliquer des conditions légales ou des mesures techniques qui restreindraient légalement autrui à utiliser l'œuvre dans les conditions décrites par la licence.

Unix/Linux utilisateur

Plan du cours

- Histoire des systèmes UNIX et GNU/Linux
- Prise en main
  - Shell et ligne de commande
  - Exploration des interfaces graphiques
- Arborescence du système de fichier
  - Arborescence standard
  - Examen du stockage
  - Manipulation des fichiers
  - Contrôle des droits d'accès
  - Éditeurs de textes, vi
  - Outils de recherche de fichiers
- Rôles du Shell
  - Paramètrage des sessions
  - Redirections d'entrées/sorties
  - Substitution d'arguments
  - Environnement
  - Scripts simples

Unix/Linux utilisateur

Plan du cours

- Outils et filtres
  - Tubes
  - Expressions régulières
  - grep et sed
  - Autres outils
- Tâches et processus
  - Exécution en arrière plan
  - Processus et démons
  - Examen des processus
  - Envoi de signaux
- Scripts Shell
  - Conditions et boucles
  - Tests
  - Interaction utilisateur

Unix/Linux utilisateur

Plan du cours

- Bases d'administration système
  - Privilège d'administration système
  - Gestion des utilisateurs et groupes
  - Installation et mise à jour de composants
- UNIX/Linux en réseau
  - Bases de TCP/IP
  - Connexion à distance
  - Transferts de fichiers
  - Services réseaux

Définition d'un système d'exploitation

Un logiciel qui présente à l'utilisateur et au dévoloppeur une machine plus simple plus générique que la machine physique.

- Au lieu de lire bit par bit, octet par octet une bande, une carte, un disque on accède aux fichiers par leur nom, leur chemin d'accès
- Des périphériques différents sont présentés de façon homogène : disque dur magnétique, disque SSD, disquette ; port série, carte son, port parallèle, ...

Cf. Andrew TANENBAUM : Principe et Implémentation des Systèmes d'Exploitation (version plus récente : *Modern Operating Systems*)

Avantage/Inconvénients

- Simplifient la programmation énormement et simplifient l'interaction entre programmes
- Consommation de ressources (temps, mémoire, disque)

Histoire des systèmes d'exploitation

Systèmes historiques (années soixantes)

- CTSS *(Compatible Time Sharing System)*, Tenex, ...
- Multics : projet ambitieux impliquant les grands industriels de l'époque (IBM, Honeywell, AT&T, General Electric, ...)
  - Très sécurisé
  - Multi-utilisateurs
  - Multitâche
- Le développement de Multics s'enlise
  - Le language PL/I supposé être utilisé prend du retard
  - Il est trop ambiteux, trop complexe
  - Son développement est organisé de façon bureaucratique, par des sociétés concurrentes

D'UNICS à UNIX

AT&T (American Telegraph & Telecom), Bell Laboratories

- Ken Thompson récupère un PDP-7 de DEC qui n'est pas utilisé
- En quelque jours il développe en langage machine un début de système d'exploitation
  - Multitâche
  - Système de fichier
  - Shell (ligne de commande)
  - Il l'appelle UNICS par dérision vs. MULTICS
  - Principe de conception KISS *(Keep it Simple Stupid)*
- Il éveille l'intérêt des autres membres du Labo (Kernighan, Ritchie, ...)

Un système portable, puissant et simple

UNIX

- L'équipe réécrit UNICS le renomme UNIX
- Sur un mini-ordinateur, le DEC PDP-11
- Création du langage C (dérivé du langage B, qui dérive de BCPL)
- Réécriture en C du noyau, du Shell, des utilitaires
- Multitâche, multi-utilisateur, sécurité sur les ressources (propriétaire, droits d'accès)
- Système de fichier homogène (données, programmes, périphériques)

Dennis Ritchie et Ken Thompson (CC BY-SA 2.0 Wikipedia)

Les seventies

AT&T utilise le système en interne

- Pour la rédaction de brevets
- Pour le développement de logiciels
- AT&T n'a pas le droit de le vendre (loi anti-trust)
- AT&T fournit le code d'UNIX pour une somme symbolique à qui le demande
- Un article est publié dans la presse scientifique : *UNIX Time-Sharing System*
- https://www.bell-labs.com/usr/dmr/www/retro.pdf

UNIX est porté sur d'autres architectures

- Interdata pour commencer, plus tard VAX
- Le compilateur C est portable

UNIX sort du laboratoire

Des universités se procurent rapidement UNIX

- *University of California, Berkeley* en particulier
- Cambridge au Royaume Uni, Sydney en Australie
- Elles ajoutent des fonctionnalités (TCP/IP !), des utilitaires
- Elle portent UNIX sur d'autres architectures

Les années quatre-vingts

Le monopole d'AT&T saute...

- AT&T peut vendre UNIX dans l'industrie
- Surtout à partir d'UNIX System V Release III (ex. Microsoft Xenix)
- Berkeley continue à proposer UNIX (BSD) courtoisement
- Procès intenté par AT&T...

UNIX se répand dans l'industrie

- Un nouveau type de machine : les stations de travail
- SUN Microsystems et autres...
- Ils adoptent naturellement UNIX
- Certains reprennent BSD, d'autres achètent une licence AT&T

Fondamentaux d'UNIX

Unix is user-friendly — it's just choosy about who its friends are.
(Anonyme)

Pour les utilisateurs et les développeurs

- Nombreux utilitaires qui font une chose et le font bien
- Shell qui permet de les faire communiquer entre eux (possible parce que on a le multitâche et des Entrées/Sorties simples)
- Système de fichier hiérarchique et unique qui contient données, programmes mais aussi des fichiers « spéciaux » qui correspondent à des périphériques

Illustration

Documentaire de 1982, AT&T Archives : *The UNIX Operating System*
https://www.youtube.com/watch?v=tc4ROCJYbm0

GNU ?

GNU : GNU's Not UNIX

- Au début des 80s, Richard Stallman, ingénieur au MIT, tombe sur un bug dans un pilote d'imprimante
- Il cherche les sources pour corriger... introuvables
- Un collègue les lui fournit mais il prévient : interdit de distribuer les sources ou même le correctif
- Stallman est furieux. Il démissionne, il lance le projet GNU
- Écrire un OS complet + applications librement distribuables
  0. Droit d'usage sans licence pour toute domaine
  1. Droit d'étudier et de modifier le programme, ceci implique l'accès au code source
  2. Droit de distribuer le programme
  3. Droit de distribuer des versions modifiées du programme
- 1991 : Il est fonctionnel
  - Compilateur (GCC), Bash, commandes UNIX usuelles, ...
  - Manque le noyau, Hurd, il prend du retard
- En 91, accord en justice : Berkeley a le droit de diffuser BSD

Pendant ce temps (1991) à Helsinki, en Finlande

Un étudiant en licence d'informatique, Linus TORVALDS, aquiert un PC 386, et il veut un vrai OS dessus...

Mais...

- Il découvre Minix fournit avec l'ouvrage d'Andrew Tanenbaum, c'est un quasi-UNIX mais... incomplet, pas beaucoup de pilotes et il est propriétaire (il faut avoir le livre pour l'utiliser, il est interdit de distribuer des versions modifiés)
- Il décide d'écrire son propre noyau en utilisant Minix + GNU comme environnement de développement au bout d'un an plus besoin de Minix et Linus le publie sur Internet et choisi la licence GPL (GNU)

Collaboration et Internet

Software is like sex: it's better when it's free. — Linus Torvalds

De Freax à Linux

- Il se répand très vite (et se développe avec beaucoup de contributeurs) dans le monde universitaire
- GNU/Linux : un environnement complet, libre, autonome
- The Story of Linux (2011) : https://www.youtube.com/watch?v=5ocq6_3-nEw
- How Linux is Built : https://www.youtube.com/watch?v=yVpbFMhOAwE
- Nom de code Linux (2002, 1h) : https://www.youtube.com/watch?v=79_IMeks4wY

Linus Torvalds et Richard Stallman

Étapes de l'industrialisation de Linux (199x - 1999 - 2020)

Distributions de GNU/Linux

- Pionniers : Yggdrasil, Slackware, ...
- Puis Debian GNU/Linux, Red Hat
- Mandrake/Mandriva (FR), SuSE (DE)
- Ubuntu (variante de Debian), CentOS, Fedora, Rocky Linux
- Aujourd'hui, 4 familles (les deux premières représentent 90% du parc)
  - Famille Debian (Debian, Ubuntu, Mint, etc.)
  - Famille RedHat (RHEL, CentOS, Fedora, Rocky, ...)
  - Plus marginal : SuSE/OpenSuSE
  - À la pointe : Gentoo, ArchLinux, Alpine, ...
- Beaucoup de choses dans l'informatique embarquée et temps réel (Yocto, Buildroot, Android, etc.)

Les autres UNIX...

Très synthétiquement...

- L'histoire d'UNIX est foisonnante : https://www.levenez.com/
- GNU/Linux a finit par dominer le marché, il reste en gros :
- Les BSD libres (FreeBSD, OpenBSD, NetBSD)
- macOS et iOS
- IBM AIX, HP/UX, Oracle (ex-SUN) Solaris, ...
- Linux est proposé par Microsoft dans Azure, mais aussi dans Windows 10+ : *Windows Subsytem for Linux* (WSL)

Architecture

Espace Noyau

- Le noyau *(kernel)* accède aux périphériques
- Il contrôle aussi l'accès au temps CPU et à la mémoire
- Il fonctionne dans le mode « superviseur » du processeur
- L'espace utilisateur sollicite des services via des *appels systèmes* peu nombreux et simples
- Simple – efficace – fiable

Espace utilisateur

- Tout le reste
- Les utilitaires et applications invoquent des appels systèmes
- Souvent par l'intermédiaire de bibliothèques (libc, etc.)
- Le Shell (ligne de commande) est l'interface privilégiée
- Ce qui n'empêche pas les interfaces graphiques d'être très présentes (X11, Wayland, CDE, macOS, Gnome, KDE, etc.)

Prise en main

Connexion

- Il faut s'authentifier pour ouvrir une session
- Vous arrivez sur un bureau graphique ou une session Shell
- Vous pouvez ouvrir un terminal pour accéder à un Shell à partir du bureau

Interfaces graphiques

- Il en existe de nombreuses : Gnome, KDE, LXDE, ...
- Gnome est la plus répandue

Activité : exploration de Gnome

- Personnalisation du bureau
- Navigation dans le système de fichiers
- Recherche d'applications
- Création de fichiers texte
- Ouverture d'un terminal

Bases de la ligne de commande

Le(s) Shell(s)

- Il en existe de nombreux : Bash, zsh, ksh, csh, fish, ...
- Il est recommandé d'utiliser un Shell de type Bourne
  - Bash (Bourne Again Shell)
  - zsh, ksh, fish
- La famille des *« C-Shells »* est peu recommandée
- *Csh Programming Considered Harmful* :
  https://www-uxsup.csx.cam.ac.uk/misc/csh.html

Syntaxe

Interface en ligne de commande

- Le Shell présente une invite (*prompt*), généralement :
  - Qui vous êtes *(whoami)*
  - Sur quel système *(hostname)*
  - Quel est le répertoire de travail *(pwd)*
  - Êtes-vous *root (super user)* ? $ (non) ou # (oui)
- Il attends ensuite votre saisie :
  - Suite d'éléments séparés par espace
  - D'abord la commande
  - Puis ses éventuels arguments, toujours séparés par espace(s)

Dans le cours nous représenterons l'invite par le simple caractère $

```
$ whoami
$ uname
$ uname -a
```

Documentation en ligne !

Il y a beaucoup de commandes !

- UNIX est un environnement logiciel riche, il y a beaucoup d'outils puissants
- Il ne s'agit pas de retenir tous ces détails !

Manuel en ligne

- Un manuel est fourni : pages *man* (et parfois commande *info*)
- `man nom_de_page` : montre la première page de ce nom dans le manuel
- Noms de commandes, de fonctions système, de configuration, de concepts
- Le manuel est organisé en sections : `man man`
- On peut faire : `man section nom_de_page`
- Les commandes `apropos` et `whereis` permettent de faire des recherches

Types de commande et arguments

Types de commandes

- Commandes internes : exécutées directement par le Shell
  - Par nécessité : `cd`, `export`, `type`, `help`, ...
  - Pour l'optimisation ou la simplicité : `echo`, `pwd`, ...
- Commandes externes : lancement de programmes
  - Recherchées dans une liste de répertoires : `echo $PATH`
  - Gestion des fichiers et répertoires : `ls`, `mkdir`, ...
  - Accès aux données : `cat`, `less`, ...

Types d'arguments

- Le ou les objet concerné(s) : fichier, répertoire, nom de système, ...
  - Suite de caractères
  - On utilise `"..."` ou `'...'` si elle contient des espaces
- Options modifiant le comportement de la commande
  - Forme courte : tiret puis un ou plusieurs caractères
  - Forme longue : double tiret puis des termes

Exemples

Lister des fichier du répertoire de travail ou d'ailleurs

```
$ pwd
$ ls
$ ls /etc
$ ls -l
$ ls -a
$ ls --all
$ /usr/bin/ls -ld /etc
```

- Que font ces options ? Consultons la documentation : `man ls`
- L'ordre des options n'importe généralement pas
  - `ls -l --all` ou `ls --all -l`
- On peut regrouper les options courtes
  - `ls -lah` au lieu de `ls -l -a -h`

Arborescence du système de fichiers

Arborescence

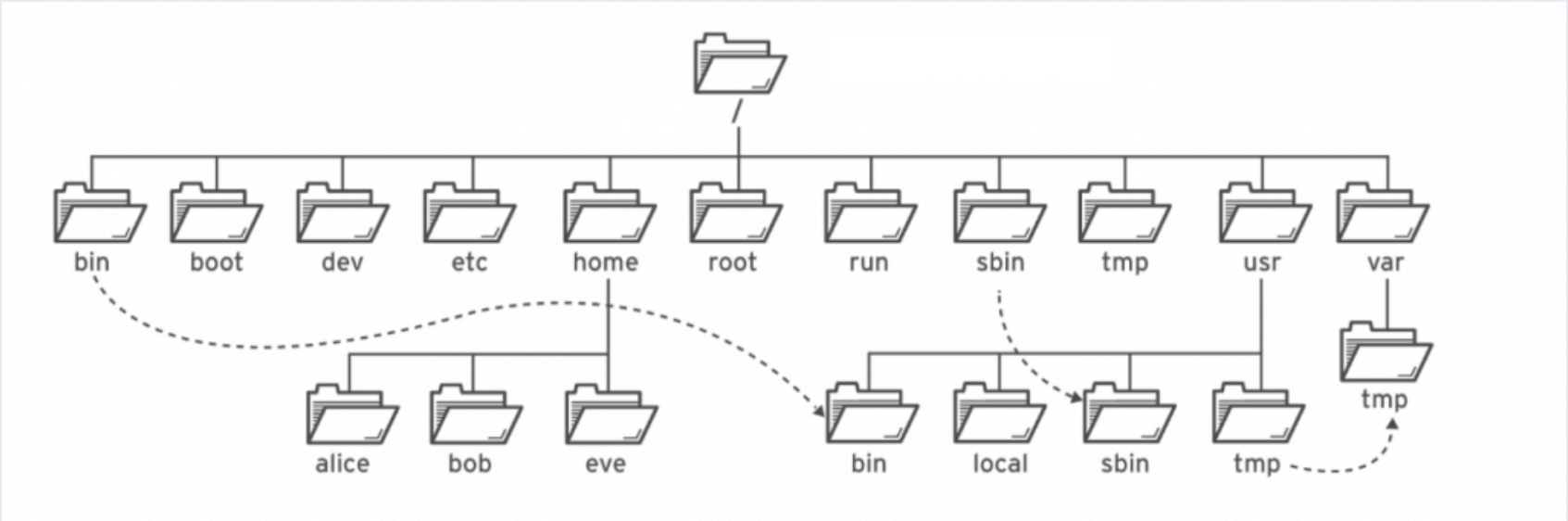
- Le système présente une arborescence unique
  - Pas d'unités comme `C:`, `D:`, ...
  - Cependant des répertoires peuvent correspondre à des volumes différents (partitions, volumes logiques, partages réseau)
- La racine du système de fichier est notée `/`
- On indique à la suite répertoires, sous-répertoires, fichiers : `/usr/bin/file`, `/etc/hosts`

Examen de l'organisation du stockage

```
$ lsblk
$ df
$ df -h
$ mount
```

Arborescence standard

D'UNIX à GNU/Linux une organisation semblable



Spécificités de GNU/Linux

- `/bin` et `/lib` sont dans `/usr`
- Les répertoires personnels de connexion sont dans `/home`
- `/boot` contient ce qui est nécessaire au démarrage (bootloader, noyau, ...)
- `man hier` : description de l'arborescence

Chemins d'accès

Comment référencer un élément de cette arborescence ?

- Fichier de données, programme, bibliothèque, ...
- Fichier « spécial » : périphérique, canal de communication
- Répertoire :
  - Entrée référençant une liste de noms
  - ... noms de fichiers ou de répertoires

Notion de base qu'il faut bien maîtriser

- Pour fournir un argument à une commnande
- Dans une application
- Dans un fichier de configuration
- Comme presque tout est fichier sous UNIX c'est un point fondamental !

Chemins absolus et relatifs

Chemins absolus

- **Débutent par /**
  - Indiquent la voie à suivre **à partir de la racine**
  - Puis des noms séparés par /
- Puisque l'on part de la racine, ils désignent la même ressource quel que soit le contexte (répertoire de travail)

```
$ ls /etc
$ ls -l /etc/hosts
$ ls /home/joe/Documents
```

Exemples

- Traduire en chemin absolu : « le fichier nommé *file* situé dans le répertoire *bin* situé dans le répertoire *usr* situé à la racine »
- De même pour : « Le répertoire nommé *journal* situé dans le répertoire *log* situé dans le répertoire *var* situé à la racine »

Chemins absolus et relatifs

Chemins relatifs

- **Ne** débutent **pas** par /
- Généralement relatifs au répertoire de travail
  - Indiqué dans l'invite du Shell (~ représente « chez vous », votre répertoire de connexion, généralement dans `/home`)
  - On peut afficher le chemin absolu vers le répertoire de travail avec la commande `pwd` *(print working directory)*

Se déplacer dans l'arborescence

Comment changer de répertoire de travail ?

- Commande interne du Shell : `cd` *(change directory)*
- Sans argument vous ramène à votre répertoire de connexion
- Accepte comme argument un chemin absolu ou relatif

```
$ cd /home ; ls
$ cd /lib ; ls
$ cd modules ; ls
```

Naviguer dans le système de fichier

- Il est *fondamental* de toujours savoir « où l'on est »
- La même commande peut avoir des effets différents selon dans quel répertoire on se trouve

Répertoires spéciaux . et ..

- Dans chaque répertoire il existe deux entrées particulières
  - `.` : mène au répertoire lui-même
  - `..` : mène au répertoire de niveau supérieur (parent)
- Ainsi ces chemins *relatifs* sont possibles :
  - `../linus/Documents/proc.txt`
  - `../../etc/hosts`
  - et mènent bien à des fichiers ou non selon le contexte (répertoire de travail)

Indiquez les chemins relatifs des exemples précédents en supposant que votre répertoire de travail est votre répertoire de connexion

Commandes de manipulation de fichiers

Opérations courantes sur les fichiers

- Copier : `cp`
- Déplacer ou renommer : `mv`
- Supprimer : `rm`
- Créer ou supprimer un répertoire : `mkdir`, `rmdir`

Comportements homogènes

- Toutes ces commandes attendent un ou plusieurs chemins absolus ou relatifs
- Des commandes similaires fonctionnent de façon similaire (copier ou déplacer par exemple)
- Ces commandes acceptent des options en argument, typiquement :
  - `-r` : récursif, `-f` : force (dangereux !)
  - `-i` : interactif
- Utilisez `man` ou l'option `--help` pour les retrouver

Création de fichiers et répertoires

Création de fichiers

- Pour s'exercer il est commode de créer des fichiers
- La commande `touch` crée un fichier vide s'il le chemin n'existe pas
- Si le fichier ou répertoire existe sa date de modification est mise à jour
- Utile pour les développeurs et les administrateurs

Création de répertoire

- La commande `mkdir` crée un répertoire
  - Si le nom n'existe pas déjà
  - Si vous avez les permissions nécessaires
- La commande `rmdir` supprime un répertoire
  - À condition qu'il soit vide (hors `.` et `..`)
  - Si vous avez les permissions nécessaires

Exemples

```
$ touch tplinux
$ touch tpunix
$ mkdir TP
$ touch TP/notes.txt ; touch TP/test.txt
$ ls -l TP
$ mkdir TP/New
$ ls -l TP
$ rmdir TP/New
$ ls -l TP ; ls -la TP
```

Des fichiers « cachés » ?

- Par convention `ls`, sans l'option `-a` ou `--all` ne montre pas les fichiers dont le nom commence par un point `.`
- Commode pour ne pas voir `.` et `..` qui sont toujours présents
- Utilisé aussi pour certains fichiers de configuration
- Examinez votre répertoire de connexion !

Copier des fichiers ou répertoires

Si La destination est un chemin de fichier (existant ou non)

`cp source destination`
La destination sera créée ou écrasée

Si La destination est un chemin de répertoire qui existe

`cp source [source2 ...] répertoire`
Plusieurs fichiers peuvent ainsi être copiés.

Àjouter l'option `-r` si l'une des sources est un répertoire !

- Copiez le fichier `tplinux` vers un fichier `tplinux.bak` dans votre répertoire `Documents`
- Copiez les fichiers du répertoire `TP` dans le répertoire `/tmp`
- Vérifiez avec `ls`
- Supprimez seulement les fichiers ici créés avec `rm`

Déplacer ou renommer des fichiers ou répertoires

Si La destination est un chemin de fichier (existant ou non)

`mv source destination`

- La destination sera créée ou écrasée
- Il peut s'agir d'un renommage (même répertoire) ou d'un déplacement (répertoires différents)

Si La destination est un chemin de répertoire qui existe

`mv source1 [source2 ...] répertoire`

Plusieurs fichiers peuvent ainsi être déplacés

- Déplacez le fichier `tplinux` vers un fichier `tplinux.bak` dans votre répertoire `Documents`, renommez `tpunix` en `TPUnix`
- Déplacez les fichiers du répertoire `TP` dans le répertoire `/tmp`
- Vérifiez avec `ls`, puis déplacez tous ces fichiers vers leurs emplacements initiaux

Supprimer des fichiers ou répertoires

Suppression d'une entrée de répertoire

`rm chemin [chemin2 ...]`

- Supprime un ou plusieurs fichiers
- Refuse d'agir sur un répertoire
  - Sauf si option `-r` (récursif) : dangereux !

Il est plus sûr de supprimer d'abord les entrées, puis les répertoires vides avec `rmdir`

- Créez un répertoire `Conf` dans le répertoire `TP`
- Copiez-y les fichiers `hosts`, `services` et `passwd` qui sont dans le répertoire `/etc`
- Supprimez le répertoire `Conf` avec `rm -r`
- Recommencez puis cette fois utilisez `rmdir` pour supprimer le répertoire `Conf`

Organisation interne du stockage

Qu'est-ce qu'un nom de fichier ou de répertoire ?

- C'est une entrée elle-même présente dans un répertoire
- Qui permet d'accéder aux informations de ces fichiers

Principe des systèmes de fichiers UNIX

- Les informations d'un fichier, sauf son nom, sont référencées par un numéro d'i-nœud *(inode)*
  - Contenu du fichier (blocs de données)
  - Liste de noms/numéro d'i-nœud pour un répertoire
  - Méta-données (propriétaire, droits, dates, etc.)
- Le nom n'est pas une propriété du fichier mais une entrée dans le répertoire concerné, associée à un i-nœud

Plusieurs noms pour un *même* fichier ?

- Il est donc possible d'avoir plusieurs noms pour un unique fichier, dans un ou plusieurs répertoire(s)

Liens physiques (ou liens « durs »)

Examiner les numéro d'i-nœuds et le nombre de liens

```
$ ls -li
```

- Montre le numéro d'i-nœud de chaque entrée
- Le troisième champ est le nombre de liens (« noms ») de chaque entrée

Créer de nouveaux liens (« noms »)

```
$ ln fichier nouveau_nom
$ ln fichier [fichier2 ...] répertoire_existant
```

Peut être utile, mais...

- Un fichier n'est jamais supprimé si son compteur de liens est non nul
- Peut servir de protection contre l'effacement involontaire
- Mais il peut devenir difficile de retrouver tous ces noms !

Liens symboliques

Plus commode

- Pour donner divers accès à un même contenu il est plus commode de créer des liens *symboliques*
- Un type spécial de fichiers (« `l` ») qui référencent un chemin absolu ou relatif (à la position du lien) vers un autre fichier ou répertoire
- La destination peut très bien ne pas exister !

Création de liens symboliques

```
ln -s chemin nouveau
ln -s chemin [chemin2 ...] répertoire
```

- Créez des liens symboliques vers divers fichiers personnels et système dans un nouveau répertoire `Symb`
- Examinez ce qui se passe selon que vous copiez ou déplacez ces liens dans votre répertoire de connexion

Droits d'accès

Permissions UNIX de base

- Chaque fichier ou répertoire est associé à un propriétaire et un groupe
- Trois modes d'accès peuvent être spécifiés :
  - `r` (Read) : lecture
  - `w` (Write) : écriture
  - `x` (eXec) : exécution
- Pour un répertoire `r` signifie lister les noms présents, `x` signifie traverser le répertoire (accéder à un sous-répertoire par exemple), et `w` y créer ou effacer un nom

3x3 = 9

- Un utilisateur est soit :
  - Propriétaire du fichier
  - Membre du groupe propriétaire du fichier
  - Autre

Examen des droits d'accès

La commande `ls -l` nous montre en premier :

- Le type du fichier
  - `-` fichier « normal »
  - `d` répertoire *(directory)*
  - `l` lien *symbolique*
  - `c`, `b`, `p`, `s`, ... autre (périphériques, tubes nommés, sockets)
- Nous nous intéressons surtout aux fichiers « normaux », répertoires et liens symboliques dans ce cours

... puis les droits

- 3 fois `rwx`
  - Si la lettre est présente : droit permis
  - Si la lettre est absente (`-`) : droit non accordé
- La commande `stat` nous montre aussi ces informations

Modification des droits

Notation symbolique

- `rwx` : droits à accorder ou non...
- `u`, `g`, `o` : propriétaire *(user)*, groupe, autre *(other)*
- `a` ou *vide* : synonyme de `ugo` *(all)*

Commande `chmod`

- On décrit d'abord les changement souhaités
- Pour qui (ugoa) et comment :
  - `+` : passe à **oui**
  - `-` : passe à **non**
  - `=` : spécifie **exactement** (passe les droits non indiqués à **non**)
- et sur quels accès : `r`, `w`, `x`

```
chmod spec1[,spec2 ...] chemin [chemin2 ...]
```

Il faut être propriétaire d'un fichier (ou *root*) pour modifier ses droits

Exemples de spécifications

- `ug+r,o-r` : permet la lecture au proprétaire et groupe, pas aux autres
- `u+x` : permet l'exécution au propriétaire
- `+rx` ou `a+rx` : permet lecture et exécution à tout le monde
- `u=rwx,go=rx` : tous droits pour le propriétaire, lecture et exécution pour groupe et autres (et *pas* écriture)

À vous. . .

- Créez un script Shell minimal : `echo "echo Hello" > script`
- Vérifiez avec la commande `cat` que vous pouvez lire son contenu
- Tentez de l'exécuter : `./script` ; est-ce permis ?
- Ajoutez le droit d'exécution au fichier ; cela fonctionne-t-il ?
- Enlevez tout droit de lecture et écriture sur le fichier ; testez
- Tentez d'y écrire : `echo "echo Coucou" > script`
- Rétablissez les droits de lecture et écriture ; testez

Expérimentez afin de vérifier qu'il est parfaitement possible d'effacer un fichier sur lequel on a aucun droit dès lors que l'on a le droit d'écriture sur le répertoire

Notation octale

Autre notation basé sur le binaire

- `rwx` : trois bits positionnés à 0 (non) ou 1 (oui)
- De 000 à 111, soit de 0 à 7 (chiffre en base 8)
- En additionnant : r=4 ; w=2 ; x=1
- Donc : `rwxr-x---` se note 750
- Notation acceptée par `chmod` et visible avec `stat`

- Convertissez en notation `rwx` tous les modes de 0 à 7
- Lesquels ont du sens selon vous ?
- Reprenez les étapes du transparent précédent avec `chmod` en utilisant la notation octale
- Consultez la page de manuel `umask` et l'aide de la commande Shell du même nom : `man umask` et `help umask`

Autres droits et autres possibilités

Il existe trois autres droits :

- *setuid* : `u+s` : une commande exécutable disposera des permissions du propriétaire au lieu de ceux de l'utilisateur qui le lance (dangereux ! à réserver à des éléments systèmes)
- *setgid* : `g+s` : pour un répertoire. Si un fichier y est créé il appartiendra au groupe du répertoire et non au groupe primaire de l'utilisateur (pour accès partagé à un répertoire)
- *sticky* : `+t` : dans le répertoire conerné le droit d'effacement devient réservé au propriétaire du fichier (cas des répertoires où tout le monde peut écrire, comme `/tmp`)

Au delà des droits UNIX historiques

- Ce modèle est simple mais couvre 99% des besoins
- Pour plus de souplesse : les ACLs *(Access Control Lists)*
- Commandes `getfacl` et `setfacl`

Édition de texte

Sous UNIX presque tout est fichier, et beaucoup de fichiers sont des fichiers textes.

- Configuration personnelle de son bureau, de son Shell
- Configuration du système
- Code sources de logiciels, documentation, etc.

Quel éditeur adopter ?

- Beaucoup de distributions de GNU/Linux proposent un éditeur minimal en mode console : `nano`
- `Emacs` est populaire chez les développeurs
- En mode graphique : `gedit` et bien d'autres !
- Un incontournable : `vi` (prononcez *vi aïe !*)

VI(sual editor) et VIM

Histoire

- Les premiers éditeurs d'UNIX n'étaient pas « pleine page »
- Édition d'une ligne à la fois... Sans voir plus du document
- Bill Joy (qui créera aussi SUN Microsystems) conçoit `vi` en 1977 à Berkeley

VI iMproved

- Une version réécrite et plus confortable populaire sous GNU/Linux
- Disponible sous tous les systèmes, y compris macOS et MS Windows

VI(m) en résumé

Mode commande

- Mode initial de lancement
- Permet de supprimer (x ou d), copier (yy), coller (p), se déplacer
- Permet de passer en mode d'insertion (i ou a ou o)
- Recherche de texte : / (division)
- Permet de passer en mode ligne de commande (touche deux points « : »)

Mode d'insertion

- La saisie de texte s'insère dans le document
- Echap/ESC permet de revenir en mode commande

Mode ligne de commande

- `:wq` : sauvegarder et quitter
- `:q!` : quitter sans sauvegarder

Apprendre vi(m) ?

- `vi` est beaucoup plus puissant que `nano` ou autres
- Il est présent sur tout UNIX raisonnablement récent
- C'est l'outil fétiche des administrateurs avertis

Cette puissance à un prix...

- Il faut un peu de temps et de pratique pour bien en profiter
- Nous n'avons pas nécessairement le temps pendant ce cours
- Vous pouvez très bien être allergique à `vi`, certains professionnels le sont : ils ont tort :-)

On peut s'initier

- `vim` fournit `vimtutor` un tutorial interactif dans plusieurs langues pour le prendre en main
- Lancez `vimtutor` dans un terminal...

Rechercher dans l'arborescence

La commande `find` est le « couteau suisse » de la recherche de fichiers

- Spécifications de critères : noms, tailles, etc.
- Recherche récursive dans l'arborescence
- Beaucoup de possibilités : `man find`

```
$ find /usr -name 'z*p'
```

Si *locate* (ou *mlocate*) est installé, une base de données des noms de fichiers est mise à jour quotidiennement et peut s'exploiter avec la commmande `locate`.

Démo : usage de `locate` et mise à jour de la base des noms. Exemples d'utilisation de `find`.

Paramètrage du Shell

Fichiers lus au démarrage

- Shells Bourne ici (Bash, zsh, ksh, ...)
- Quand un Shell se lance il parcourt (il « source ») plusieurs fichiers scripts
- Selon le Shell les noms de ces fichiers diffèrent (`.profile` à l'origine du Shell Bourne)
- Nous allons donner les détails pour *Bash*

- `/etc/bash...` : pour tous les utilisateurs
- `~/.bash_login` : pour un Shell de connexion
- `~/.bashrc` : pour tout Shell (Terminal...)

Personnaliser notre configuration

- Ouvrez votre `.bashrc` avec un éditeur de texte
- Ajoutez les lignes suivantes :

```
HISTSIZE=5000
HISTFILESIZE=10000
alias ll='ls -l'
alias la='ls -a'
echo "Bonjour $USER !"
```

- Consultez le manuel de *Bash* pour en savoir plus sur les possibilités de personalisation... Vaste sujet !
- Les distributions de GNU/Linux fournissent un environnement initial bien pensé qu'il est peu nécessaire d'adapter

Redirection des sorties

Shells et Entrées/Sorties

- Le Shell communique à travers un terminal
- Il y a trois flux de communications définis :
  - *stdin* ou 0 : entrée standard
  - *stdout* ou 1 : sortie standard
  - *stderr* ou 2 : sortie pour messages d'erreur ou avertissements
- Quand vous exécutez une commande elle hérite de ces trois canaux et les utilise pour communiquer (lire et écrire)

Exemple : `cat` lit les fichiers passés en arguments s'il y en a et *sinon* lit l'entrée standard (qui se termine au clavier par CTRL+D)

```
$ cat /etc/networks /etc/hosts
$ cat
Hello
Hello
^D
```

Redirection de l'entrée et des sorties

Redirection entrée standard

```
$ cat < /etc/hosts
```

Redirection de la sortie standard ou d'erreur

```
$ ls -l > liste
$ cat liste
$ ls -l nothere 2> erreur
$ cat erreur
```

Par défaut le fichier est vidé si il existe

Redirection en ajout à la fin de la sortie

```
$ ls -l /usr /bin /lib >> liste
$ cat liste
```

Pour faire disparaître une des sorties : redirigez-la vers le fichier spécial de périphérique `/dev/null` !

À vous...

Exécutez à la suite des commandes qui enregistrent dans un fichier `stockage.txt` les informations issues des commandes `mount`, `lsblk` et `df -h`. Examinez le fichier produit.

Note : la commande `getent` permet d'obtenir (entre autres) des informations sur les comptes et groupes du système.

Utilisez `cat` pour envoyer dans un fichier `output.txt` les contenus des fichiers `/etc/timezone`, `/etc/shells`. Ensuite ajoutez à ce fichiers le contenu des sorties des commandes `getent passwd` et `getent group`. Examinez le fichier produit.

Substitution d'arguments : jokers

Traiter plusieurs fichiers à la fois

- Le Shell propose des caractères spéciaux (« jokers ») pour évoquer plusieurs noms de fichiers (ou chemins) existants
- `*` : n'importe quelle séquence de caractère
- `?` : un caractère quelconque
- `[aeiou]` : un caractère dans une liste
- `[a-z]` : un caractère dans un intervalle
- `[!...]` : un caractère pas dans la liste ou l'intervalle

```
$ ls -l *.txt
$ ls -ld /usr/[a-m]*
$ ls -ld /usr/[!a-m]*
$ ls -ld /usr/[n-z]*
$ file *
```

Application à la manipulation de fichiers

Créez avec `touch` des fichiers nommés `data01`, `data02`, ..., `data12`. Utilisez `mv` pour déplacer `data01` jusqu'à `data07` uniquement dans `Documents`. Vérifiez.

Utilisez `rm` pour supprimer tout fichier de la forme `data1x` où `x` pourrait valoir de 0 à 9. Vérifiez.

Quelles commandes dans `/usr/bin` commencent par un z suivi de trois caractères quelconques et se terminent par p ?

Que se passe-t-il si aucun nom ne correspond à une expression ? Réessayez après avoir exécuté `shopt -s failglob`. Pensez-vous que ce comportement est préférable ?

Environnement

Une commande s'exécute dans un *environnement*

- Un ensemble de variables ayant des valeurs
- Paramètre le comportement du Shell et de bien d'autres commandes et applications
- Permet de récuperer des informations précieuses : nom du système, nature du système d'exploitation, identifiant ulisateur, ...

```
$ env
$ echo $LANG
$ man ls
$ LANG=C man ls
$ man ls
$ export LANG=C
$ man ls
```

Scripts simples

Un script est un fichier texte

- Contenant des commandes Shell
- Bonne pratique : débuter en indiquant le Shell concerné (ligne *shebang*)
- On peut y créer des variables quelconques
- Qui seront d'environnement (héritées) si vous utiliser `export`

```
#!/usr/bin/env bash
cours="UNIX/Linux utilisateur"
echo "Formation $cours"
echo "Vous êtes connecté à $HOSTNAME !"
read -p "Votre prénom : " prenom
echo "Bonjour $prenom !"
```

```
$ ./bonjour
```

Tubes et filtres

Nous avons vu comment rediriger la sortie standard vers un fichier et comment lire un fichier via l'entrée standard

- Les Shells UNIX proposent une possibilité encore plus puissante !
- Envoyer la sortie standard d'une commande sur l'entrée d'une autre
- Et ainsi de suite : tube *(pipe)* et *pipelines* !

```
$ getent passwd | grep /home
$ getent group | grep $USER
```

La commande `grep` est un *filtre* qui ne passe en sortie que les lignes contenant, *ici*, `/home` ou votre identifiant

Commandes grep et sed

grep est un filtre

- Filtre : commande qui lit (sauf exception) son entrée standard et envoye le résultat sur la sortie standard
- Les filtres sont conçus pour être utilisés avec des tubes

```
grep motif [fichier ...]
```

Un autre filtre : sed

Aussi expressif que `vi` ! Par exemple pour rechercher remplacer :

```
sed -e 's/motif/remplacement/[gi...]'
```

Afficher les sous-répertoires de `/usr` ainsi :

```
$ ls -d /usr/* | sed -e 's/\/usr\//Répertoire /'
```

Beaucoup d'outils, d'applications et langages de programmation acceptent la même syntaxe pour les *motifs*. Nous les illustrerons avec `grep`.

Expressions régulières

- C'est la syntaxe pour les motifs utilisable avec `grep` et autres
- Principe similaire aux *jokers* du Shell mais beaucoup plus puissant
- Utile en traitement de données, administration système, programmation, linguistique, ...
- La syntaxe étendue (moderne) est utilisable avec `grep -E` et est préférable

Caractères spéciaux d'expressions régulières

- `.` : n'importe quel caractère
- `[...]` : caractère dans une liste ou un intervalle
- `[^...]` : caractère *pas* dans une liste ou intervalle
- `*` : l'expression précédente répétée zero fois ou plus
- `+` : l'expression précédente répétée une fois ou plus
- `?` : l'expression précédente présente une fois ou absente

Expressions régulières

- ^ et $: ancres (débuts et fin de lignes)
- (...) : regroupement et étiquetage
- (chat|chien) : alternative

```
$ getent passwd | grep '^d.*'
# Liste les groupes dont vous êtes l'unique membre
# (hors groupe primaire) :
$ getent group | grep ':votrelogin$'
$ grep 'host$' /etc/hosts
# " " au lieu de ' ' pour évaluer $LOGIN
$ who | grep "^$LOGIN"
```

À vous...

Utilisez `ls` et `grep` ensembles pour lister tous les noms de liens symboliques, puis de répertoires, puis de fichiers « normaux » du répertoire courant

Examinez le contenu du fichier `mydata`. Utilisez `grep` pour extraire les lignes :

- Contenant un chiffre
- Contenant deux chiffres
- Contenant un nombre entier
- Évoquant un chien, ou un chat, ou les deux
- Les lignes contenant `data0NN`
- Les lignes contenant `data0NN` où NN va de 02 à 07
- Les lignes contenant une adresse e-mail valide (attention à faire ceci soigneusement en production, n'hésitez pas à rechercher une expression régulière validée !)

Autres outils et langages

Outre `grep` et `sed` les expressions régulières profitent à beaucoup d'outils et langages :

- *find*
- *awk*
- *Perl* et *Python*
- *Javascript*, *Java*, *C#*, ...
- Microsoft Office et LibreOffice
- Bases de données (PostgreSQL par exemple)

Autres filtres et outils

Avant de réinventer la roue...

... il vaut mieux trouver le bon outil !

Une liste (non exhaustive) à considérer :

- `sort` : trier
- `tr` : transformer/translater des caractères
- `uniq` : supprimer des doublonss
- `cut` : découper des lignes
- `column` : mettre en forme des tableaux
- `tee` : réplique un flux vers un fichier
- `cmp` et `diff` : comparent deux fichiers ou ensembles de fichiers
- ...

Tâches en avant- et arrière- plan

Plusieurs programmes ?

- UNIX et Linux sont intrinsinquement multitâches
- Beaucoup de processus tournent en parallèle sur votre système
  - Gestion des périphériques
  - Services réseaux, journaux d'activité, ... (« Démons »)
  - Composants du bureau Gnome ou autre

Shell et tâches

- Lorsque le Shell exécute une commande externe :
  - Il crée une copie de lui-même *(fork)*
  - Le « parent » se met en attente
  - Le « fils » exécute la commande *(exec)* et se termine
  - Le « parent » reprend le contrôle du terminal
- Il est possible de contrôler ce comportement

Avant-plan et arrière-plan

Contrôle des tâches

- Une commande est par défaut en avant-plan et le Shell est en attente
- En terminant la commande par & on demande au Shell de lancer une tâche en arrière-plan
- Une tâche qui veut contrôler le terminal se mettrait en attente...
- On peut contrôler plusieurs tâches :
  - `jobs` : liste des tâches du terminal courant
  - `fg` : passe une tâche au premier-plan *(foreground)*
  - `bg` : passe une tâche en arrière-plan *(background)* où elle s'exécute (si possible...)
  - ctrl+Z : « stoppe » la tâche en avant-plan

```
$ xeyes &
$ xeyes &
$ jobs
```

Processus

Tâches en cours et processus

- Les tâches contrôlées par un Shell ne sont qu'une partie des processus en cours gérées par le noyau UNIX ou Linux
- Des commandes permettent de les examiner tous ou sélectivement

```
$ ps
$ ps aux # ou ps -elf
$ pstree
$ top # q pour quitter
```

Chaque processus dispose d'un identifiant numérique unique (PID), d'un processus parent, d'un propriétaire, de divers états (activité, consommation mémoire, fichier ouverts...)
Démo : examen du répertoire `/proc`.

Envoi de signaux

Communiquer avec un processus ?

- UNIX prévoit l'envoi de *signaux* aux processus (ctrl-C, ...)
- Utile pour leur demander de relire leur configuration, s'arrêter, s'endormir, les « tuer » ...
- Liste des signaux : `kill -l`
- Envoyer un signal à un pid : `kill -TERM 88742`
- Il y a aussi `pkill` pour sélectionner finement (et `pgrep`)

Principaux signaux

- INT, TERM, QUIT : fin « gracieuse » si non bloqué
- HUP : souvent signifie « relire sa configuration »
- KILL (9) : fin sans condition (dangereux !)
- STOP et CONT : endormir et réveiller

```
$ ps aux | grep xeyes
$ pgrep xeyes
$ pkill xeyes # par défault : TERM
```

Scripts Shells

Automatiser des tâches complexes

- Le Shell nous permet d'automatiser des tâches complexes
- Déclencher certaines opérations sélectivement
- Vérifier le bon fonctionnement d'une opération
- Tester des propriétés de fichiers
- Réaliser des traitements répétitifs

Code de retour

- Quand un processus se termine il renvoie un *code*
- 0 signifie « ok », autre chose qu'il y a un problème
- Vos propres scripts peuvent se comporter ainsi
  - Terminer par `exit 0` si tout va bien
  - Une autre valeur s'il y a eu un problème

Tester une valeur de retour

Pour des opérations simples

- Opérateurs logiques « court-circuit » : `&&` (et) ; `||` (ou)
- `comm1 && comm2` : comm2 s'exécute si comm1 a réussi
- `comm1 || comm2` : comm2 s'exécute si comm1 a échoué

```
$ ls /tmp/test || echo "/tmp/test absent"
$ ls /tmp/test && echo "/tmp/test présent"
$ touch /tmp/test # et recommencer
```

Opérateur `test`

- La commande (interne ou externe) `test` permet de tester, entre autres, des propriétés de fichiers
- `test -f ...` : fichier normal existe
- `-r`, `-w`, `-x` : fichier lisible, écrivable, exécutable
- `help test` ou `man test`

À vous. . .

Exécutez une ligne de commande qui affiche *ok* si un fichier `monprog` est présent dans `/tmp` et est exécutable. Testez.
Placez cette ligne dans un script `test_monprog`, rendez-le exécutable et testez-le.

Rappel : interaction utilisateur

Pour stocker une information saisie par l'utilisateur vous pouvez utiliser `read -p message variable`. La valeur est ensuite accessible via `$variable`.
`test` permet aussi de comparer deux variables, comme texte (chaînes de caractères) ou par des comparaisons numériques. Écrivez un script qui demande un identifiant *(login)* d'utilisateur et affiche "C'est moi" si c'est. . . vous, ensuite il demande un âge et s'il est supérieur à 42 affiche "Ok boomer!".

Autre syntaxe d'exécution conditionnelle

Pour des opérations plus complexes. . .

S'il y a plus qu'une simple commande à exécuter ou non une autre forme est préférable (la clause *else* n'est pas obligatoire) :

```
if test -f /tmp/truc
then
  echo "truc est présent dans /tmp"
else
  echo "truc est absent dans /tmp"
fi
```

Avec cette syntaxe la notation `[ ... ]` (équivalente à `test`) est plus lisible :

```
if [ -f /tmp/truc ]
...
```

Réécrivons les scripts précédents avec cette forme.

Boucle `for` et substitution

`for` permet de faire parcourir une liste de valeurs à une variable :

```
for name in Ada Alan Brian Linus
do
  echo "$name est célèbre en informatique."
done
```

Substitution

Deux formes de substitution sont particulièrement utiles ici (et fonctionnent aussi avec d'autres commandes que `for`) :

- `$*` ou (mieux) `"$@"` pour la liste des arguments reçus par notre script (note : on peut aussi y accéder un par un : `$1`, `$2`, ...)
- `$( commande ... )` : la sortie de la commande devient une liste d'arguments

Substitution en liste et boucle for

```
for arg in "$@"
do
  echo "Param : $arg"
done
```

```
# que fait le who | ... cut ... ?
for user in $( who | cut -d' ' -f1 )
do
  echo "$user est connecté"
done
```

```
for pid in $( pgrep xeyes )
do
  kill -TERM $pid
done
```

À vous...

Écrivez un script qui extrait de la sortie de `getent passwd` les informations sur les utilisateurs actuellement connectées.

Écrivez un script qui accepte une suite d'identifiants d'utilisateurs *(login)* et affiche *xxx connecté* pour ceux qui sont connectés

Écrivez un script qui accept une liste de noms de machines ou d'adresses IP et lance un `ping -c 1 ...` vers chacun de ces hôtes.

Boucle `while`

Boucler sur une condition

- Exécute répétivement une séquence de commandes
- Même type de condition que `if` (état de sortie, commande `test` ou `[`, etc.)

```
while [ -f /tmp/chose ]
do
  echo "Pas de chose dans /tmp"
  sleep 1 # dort une seconde
done
echo "chose trouvée dans /tmp !"
```

Écrire un script qui attend qu'un processus nommé *xeyes* apparaisse et alors le tue (note : utiliser pgrep et kill).

Bases d'administration système et réseau

Ce n'est pas un cours d'administration système, cependant il peut être bon d'approcher le sujet pour mieux pouvoir gérer nos systèmes.

L'utilisateur root

- L'utilisateur *root* d'identifiant numérique 0 a tous les droits
- Certaines distributions proposent de lui attribuer un mot de passe
- D'autres préfèrent configurer `sudo` pour vous permettre de devenir *root* plus facilement

Changer d'identité : `su` ou `sudo` ?

- `su -` permet de lancer un Shell *root* en spécifiant le mot de passe du compte *root*
- `sudo -s` ou `sudo commande` le permet si il est configuré, en indiquant *son mot de passe à soi*

Configuration de sudo

Généralement sudo est accompagné d'une configuration où les membre d'un certain groupe on le droit de l'utiliser pour devenir *root* :

- Groupe *sudo* sous Debian, Ubuntu, ...
- Groupe *wheel* sous Red Hat, CentOS, Rocky, ...

Ubuntu (et d'autres) font en sorte que l'utilisateur créé à l'installation soit membre de ce groupe. Red Hat (etc.) le propose. Ce qui suit n'y est donc pas nécessairement nécessaire.

```
$ su -
# usermod -a -G sudo identifiant # ou wheel...
# exit
```

Se déconnecter alors du bureau pour y revenir...

```
$ sudo whoami
root
$ sudo apt install xbill
```

Installation et mise à jour

Les deux principales familles de distributions fournissent des outils d'installation et mise à jour. Tous ces outils se connectent à des *dépôts* de logiciels disponibles sur l'Internet et gèrent les dépendances.

- `apt` chez Debian, Ubuntu, ...
- `yum` (alias `dnf`) chez Red Hat, CentOS, Rocky, ...

Principes similaires

- `apt update`, `dnf check-update` : mise à jour de la liste connue
- `apt install ...`, `dnf install ...` : installation
- `apt upgrade / full-upgrade`, `dnf update` : mise à jour

Voir le manuel d'administration système pour plus de détail...

Connexion à distance

- Au départ les connexions à distance n'étaient pas chiffrées !!!
- Les outils comme `ftp`, `rsh`, `rlogin`, `rcp` ont laissé la place à SSH *(Secure SHell)*
- Protocole chiffré avec authentification à clef publique possible
- `scp` pour copier des fichiers

```
linus@helsinki:~$ ssh admin@redmond
admin@redmond:~$ exit
# \ est juste là pour passer à la ligne
# sans souci
linus@helsinki:~$ scp rapport.txt \
                      admin@redmond:Documents/
```

Démo interactive : configurer l'authentification à clef publique SSH entre nos systèmes pour nos comptes.

Configuration réseau sous GNU/Linux

Examen de la configuration courante

La commande `ip` permet d'examiner la configuration TCP/IP actuelle :

- `ip link show`, `ip addr show` : liens et adresses
- `ip route show` : table de routage
- `ping ...` et `traceroute ...` pour tester
- `cat /etc/resolv.conf` : examen de la configuration DNS (résolution de noms)

Reconfiguration

Le réseau peut être reconfiguré à travers le bureau Gnome, par des outils d'administration (`nmcli`, `nmui`) ou bien par des fichiers de configuration selon les cas.

Services réseaux habituels sous UNIX/Linux

Partage de ressources

Outre SSH qui permet d'exécuter des commandes à distance et copier des fichiers on peut partager des répertoires en réseau :

- NFS *(Network File System)* : protocole standard d'UNIX
  - Définition des partages dans `/etc/exports`
  - Montage direct ou utilisation de l'automonteur client
- SMB *(Server Message Block)* : protocole d'origine Microsoft, UNIX est compatible, y compris côté serveur avec le serveur Samba

Centralisation des comptes et groupes

- Historiquement SUN Microsystems a publié NIS *(Network Information System)*
- Il a été supplanté par LDAP *(Lightweight Directory Access Protocol)*
- Passerelles vers *Active Directory* (Microsoft)

Services réseaux applicatifs et applications

De nombreux logiciels libres fournissent des services réseaux populaires, ce sont souvent les implémentations les plus répandues et performantes

Exemples. . .

- *Apache* et *Nginx* pour le Web
- Navigateur Web *Firefox*
- *PostgreSQL*, *MySQL/MariaDB* pour les bases de données relationnelles
- *Postfix* et *Cyrus IMAP* pour le courrier électronique
- *BIND* pour le DNS
- *OpenLDAP* pour les annuaires
- `netfilter/iptables/nftables` (Linux) pour le *firewall*
- *Virtual Box*, *libvirt*, KVM et Xen pour la virtualisation et le *cloud*
- *Docker*, *LXC*, *Kubernetes*, etc. . .

Pour finir. . .

Cette promenade sous UNIX vous a plu ?

Pour aller plus loin :

- Vincent Lozano, *Unix. Pour aller plus loin avec la ligne de commande* : https://archives.framabook.org/unixpou-allerplusloinaveclalignedecommande/
- Innombrables ressources Internet, groupes, forums, . . .
- La communauté francophone : http://linuxfr.org
- Nos cursus de formations d'administrateurs systèmes, développeurs, réseau, . . .

« UNIX: Live Free or Die » (vu sur une plaque d'immatriculation aux U.S.A.)