

# Administration du système

Matthieu Moy, Grégoire Pichon

Univ. Claude Bernard Lyon 1

## Séance 4

Matthieu Moy	matthieu.moy@univ-lyon1.fr	CM + TD + TP
Grégoire Pichon	gregoire.pichon@univ-lyon1.fr	CM + TD + TP
Sylvain Brandel	sylvain.brandel@univ-lyon1.fr	TD + TP
Guillaume Damiand	guillaume.damiand@univ-lyon1.fr	TP
Laurent Lefevre	laurent.lefevre@ens-lyon.fr	TP
Frédéric Suter	frederic.suter@cc.in2p3.fr	TD + TP



# Système et administration

## Système d'exploitation (jusqu'ici)

- Savoir comment les choses sont organisées
- Pour comprendre les problèmes
  - ▶ Problèmes mémoire
  - ▶ Interblocages
- Pour utiliser les caractéristiques du système
  - ▶ Communications entre processus
  - ▶ Multi-threading
  - ▶ Gestion des ressources

# Système et administration

## Système d'exploitation (jusqu'ici)

- Savoir comment les choses sont organisées
- Pour comprendre les problèmes
  - ▶ Problèmes mémoire
  - ▶ Interblocages
- Pour utiliser les caractéristiques du système
  - ▶ Communications entre processus
  - ▶ Multi-threading
  - ▶ Gestion des ressources

## Administration (ce cours)

- Installation
- Utilisateurs
- Services



- 1 Les utilisateurs
  - Gestion des utilisateurs
  - Gestion des droits

- 2 Services, systemd

- 3 Outils de diagnostic

- 4 Installation
  - Première installation
  - Installation de logiciel
  - Gestion des paquets

- 1 Les utilisateurs
  - Gestion des utilisateurs
  - Gestion des droits

- 2 Services, systemd

- 3 Outils de diagnostic

- 4 Installation
  - Première installation
  - Installation de logiciel
  - Gestion des paquets

- 1 Les utilisateurs
  - Gestion des utilisateurs
  - Gestion des droits
- 2 Services, systemd
- 3 Outils de diagnostic
- 4 Installation
  - Première installation
  - Installation de logiciel
  - Gestion des paquets

# Utilisateurs

- Un ordinateur peut être utilisé par plusieurs « utilisateurs » :
  - ▶ Ordinateur de la fac : tous les étudiants Lyon 1 + la DSI
  - ▶ Votre ordinateur personnel : vous (quand vous travaillez) et vous (quand vous installez des logiciels ou reconfigurez le système)
  - ▶ Un « utilisateur » (informatique) n'est pas forcément une personne physique !



# (Multi-)Utilisateurs : cas d'utilisation

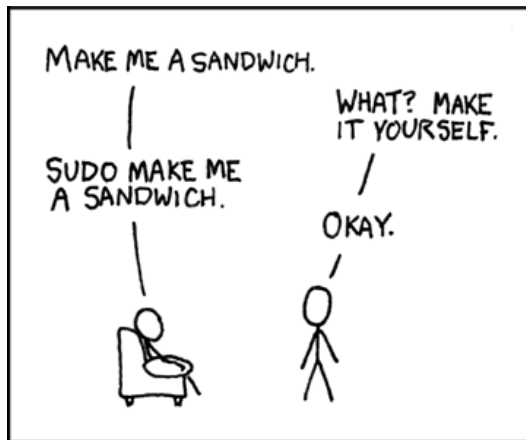
- Utilisateur d'un PC individuel (un seul utilisateur physique)
  - ↪ Séparer les tâches quotidiennes et les tâches dangereuses. Interdire aux applications classiques (e.g. navigateur web) de modifier la configuration du système.
- Utilisateurs du PC familial (quelques utilisateurs qui se font confiance)
  - ↪ Une configuration différente par utilisateur (exemple : fond d'écran différent, bookmarks du navigateur, ...)
- Utilisateurs d'un PC partagé (utilisateurs ne se faisant pas confiance a priori les uns aux autres)
  - ↪ Gestion des droits (interdiction pour un utilisateur de lire/écrire les fichiers des autres, impossibilité d'agir sur un processus appartenant à un autre utilisateur, ...).

Tous les systèmes modernes pour PC (Linux, Windows, Mac OS) proposent du multi-utilisateur.





## sudo



# Utilisateurs : administrateurs et non-privilégiés

- Certains utilisateurs ont tous les droits : les administrateurs (root sous Unix).  
~> À n'utiliser que quand on en a vraiment besoin (ne **jamais** travailler en mode admin pour des tâches du quotidien comme naviguer sur le web, compiler un programme, ...). À ne pas utiliser pour ouvrir une session graphique.
- Les autres ne peuvent faire que ce qui leur est autorisé.  
~> À utiliser tout le temps !
- Pour passer administrateur temporairement :
  - ▶ Linux : `sudo commande` ou `su` - sous Linux
  - ▶ Windows : clic droit → « run as different user », ou `runas.exe`

# Bêtises à ne pas faire ...

- La bêtise de base

- ▶ `rm -fr /` : est-ce grave ?
- ▶ `sudo rm -fr /` : est-ce grave ?

# Bêtises à ne pas faire ...

- La bêtise de base

- ▶ `rm -fr /` : est-ce grave ?
- ▶ `sudo rm -fr /` : est-ce grave ?<sup>1</sup>

---

1. en fait, non, `rm` refuse d'effacer tout `/` par défaut, mais bon ... `chmod 777 -R /` est tout aussi dangereux par exemple

# Bêtises à ne pas faire ...

- La bêtise de base
  - ▶ `rm -fr /` : est-ce grave ?
  - ▶ `sudo rm -fr /` : est-ce grave ?<sup>1</sup>
- Bêtise plus subtile :
  - ▶ `make` : je fais mon TP
  - ▶ `sudo make` : est-ce grave ?

---

1. en fait, non, `rm` refuse d'effacer tout `/` par défaut, mais bon ... `chmod 777 -R /` est tout aussi dangereux par exemple

# Bêtises à ne pas faire ...

- La bêtise de base

- ▶ `rm -fr /` : est-ce grave?
- ▶ `sudo rm -fr /` : est-ce grave?<sup>1</sup>

- Bêtise plus subtile :

- ▶ `make` : je fais mon TP
- ▶ `sudo make` : est-ce grave?

**Oui!** Cette commande va créer des fichiers qui appartiennent à root sur votre compte, et vous n'aurez plus les droits dessus après!

---

1. en fait, non, `rm` refuse d'effacer tout / par défaut, mais bon ... `chmod 777 -R /` est tout aussi dangereux par exemple

# Unix : sudo Vs su

- **su = Set User**

- ▶ Ouvre un shell root (ou toto si on fait `su toto`)
- ▶ Demande le mot de passe root
- ▶ Peut-être lancé par n'importe qui, mais il faut le mot de passe
- ▶ Variantes : `su -` (recommandé) ouvre un shell comme si on s'était loggé comme root, `su` ouvre un shell root mais garde le répertoire courant tel quel (donc root risque de faire des bêtises sur votre compte).
- ▶ Quand on a fini : terminer le shell root (`exit` ou `Control-d`).

- **sudo = {Substitute,Super} User DO**

- ▶ Commande configurable (`/etc/sudoers`)
- ▶ `sudo vi /etc/profile` : lance `vi /etc/profile` en root (retour à la normale en quittant `vi`)
- ▶ `sudo -s` : ouvre un shell (comme `su`)
- ▶ Par défaut sous Ubuntu, Mac OS X, ... :
  - ★ Demande le mot de passe de l'utilisateur courant ( $\neq$  `su`)
  - ★ Autorisation valable 15 minutes pour les prochaines commandes
  - ★ Autorisé uniquement pour les membres du groupe `admin` (attention à ne pas mettre n'importe qui)



# Informations sur un utilisateur

Un compte utilisateur est défini par :

- Un numéro (UID = User IDentifier, commande `id` pour le voir), utilisé en interne par le système
- Un nom de login
- Un répertoire personnel (`$HOME`, par défaut `/home/$LOGNAME`)
- Quelques autres méta-données (nom complet, shell par défaut, ...)

Vocabulaire : Compte Vs Session

- Compte = informations persistantes sur l'utilisateur
- Session = période de temps entre la connexion (login) et la déconnexion (logout).
- $\Rightarrow$  arrêtez de dire « je me connecte sur ma session » pour ne pas passer pour des noobs ;-).





# Utilisateurs locaux

- Les informations sur les utilisateurs sont stockées dans un fichier de la machine
  - ▶ `%systemroot%\system32\config\SAM` sous windows
  - ▶ `/etc/passwd`, `/etc/shadow` et `/etc/groups` sous linux
- Les fichiers ne contiennent pas directement les mots de passe mais leur empreinte numérique par une fonction de hachage.
  - ▶ Pour authentifier un utilisateur, le système récupère le mot de passe en clair.
  - ▶ Il utilise la même fonction de hachage et compare les résultats.
- Ces fichiers sont critiques pour le système
  - ▶ Problème des mots de passe identiques.
  - ▶ Problème des mots de passe trop simples.



# Gestion des utilisateurs locaux (Linux)

- `adduser luser` : création de l'utilisateur `luser` (entrée dans `/etc/passwd` et, création du `$HOME`, ...)
- `deluser luser` : supprime l'utilisateur `luser`.  
`deluser --remove-home luser` : idem, mais supprime aussi le `$HOME`.
- `passwd luser` : changer le mot de passe de `luser`. Demande l'ancien mot de passe sauf si lancé par `root`.
- `useradd`, `userdel` : idem, mais moins conviviales (à utiliser dans des scripts).



# En réseau

- Dans un réseau local, il est nécessaire de centraliser la gestion des utilisateurs.



# En réseau

- Dans un réseau local, il est nécessaire de centraliser la gestion des utilisateurs.
- On peut modifier les utilitaires qui accèdent aux descriptions des utilisateurs pour qu'ils contactent un serveur.
  - ▶ Ex : NIS à chaque accès, le fichier correspondant est demandé au serveur.
  - ▶ Le service utilisé pour chaque fichier est géré par le « Name Service Switch » (fichier `/etc/nsswitch.conf`).
  - ▶ Cache local.



# En réseau

- Dans un réseau local, il est nécessaire de centraliser la gestion des utilisateurs.
- On peut modifier les utilitaires qui accèdent aux descriptions des utilisateurs pour qu'ils contactent un serveur.
  - ▶ Ex : NIS à chaque accès, le fichier correspondant est demandé au serveur.
  - ▶ Le service utilisé pour chaque fichier est géré par le « Name Service Switch » (fichier `/etc/nsswitch.conf`).
  - ▶ Cache local.
- On peut déléguer une partie du travail à un serveur
  - ▶ Ex : les domaines windows
  - ▶ le PDC (contrôleur de domaine principal) fournit l'authentification
  - ▶ le reste est fait par des scripts

# En réseau

- Dans un réseau local, il est nécessaire de centraliser la gestion des utilisateurs.
- On peut modifier les utilitaires qui accèdent aux descriptions des utilisateurs pour qu'ils contactent un serveur.
  - ▶ Ex : NIS à chaque accès, le fichier correspondant est demandé au serveur.
  - ▶ Le service utilisé pour chaque fichier est géré par le « Name Service Switch » (fichier `/etc/nsswitch.conf`).
  - ▶ Cache local.
- On peut déléguer une partie du travail à un serveur
  - ▶ Ex : les domaines windows
  - ▶ le PDC (contrôleur de domaine principal) fournit l'authentification
  - ▶ le reste est fait par des scripts
- Avec ces deux méthodes les informations centralisées sont limitées.



# Annuaire

- Un annuaire est une base de données
  - ▶ Optimisée pour la lecture,
  - ▶ Pouvant contenir tout type d'information,
  - ▶ Avec une organisation hiérarchisée (arbre),
  - ▶ Permettant des recherches multiples,
  - ▶ Proposant un système d'authentification.
- Par exemple :
  - ▶ OpenLdap « Lightweight Directory Access Protocol ».
  - ▶ Active Directory qui utilise le protocole de nom ldap.

# Ldap/AD

- Les objets sont placés dans une *structure arborescente*.
- La racine de la structure est liée au domaine DNS. DC=polytech,DC=upmc,DC=fr (*DC = Domain Component*)
- chaque objet a un nom unique le *Distinguished Name* ou *DN* faisant apparaître le chemin dans l'arbre
  - ▶ OU=comptes,DC=polytech,DC=upmc,DC=fr par exemple l'entité qui rassemble tous les comptes (*OU = Organizational Unit*)
  - ▶ OU=encad,OU=comptes,DC=polytech,DC=upmc,DC=fr par exemple l'entité qui rassemble tous les enseignants
  - ▶ CN=rico,OU=encad,OU=comptes, DC=polytech,DC=upmc,DC=fr mon compte
- À chaque objet on associe des données
- Le type des données et leurs positions dans l'arbre sont fixés par des *schémas* (donc identiques entre serveur, mais adaptables).
- Les droits d'accès aux données sont gérés par des *ACL* (Access Control List)





## 1 Les utilisateurs

- Gestion des utilisateurs
- Gestion des droits

## 2 Services, systemd

## 3 Outils de diagnostic

## 4 Installation

- Première installation
- Installation de logiciel
- Gestion des paquets

# Gestion des droits

L'existence de différents utilisateurs sur une machine permet de gérer différents droits.

- Un utilisateur standard a le droit
  - ▶ D'utiliser les logiciels.
  - ▶ D'utiliser son espace de stockage (compte).
  - ▶ De lire les données partagées.
- Certains utilisateurs particuliers servent à
  - ▶ Limiter les droits des serveurs (util. apache).
  - ▶ Gérer des accès distants (Administration à distance).
  - ▶ Avoir des configurations particulières des droits particuliers (ex : oracle).
- Un utilisateur spécial a tous les droits *Administrateur* (windows) *root* (unix).



# Gestion des droits unix

- Les droits sont les droits sur les fichiers (tout est fichier).



# Gestion des droits unix

- Les droits sont les droits sur les fichiers (tout est fichier).
- Les droits de base sont :
  - ▶ `read` lecture du fichier, liste du contenu du répertoire ;
  - ▶ `write` écriture dans le fichier, ajout/suppression de fichier dans le répertoire ;
  - ▶ `execute` exécution du fichier, aller dans le répertoire *ou un sous répertoire*.
- Pour un utilisateur, un fichier est dans l'une des classes :
  - ▶ `user`, `u` : propriétaire ;
  - ▶ `group`, `g` : groupe du propriétaire ;
  - ▶ `other`, `o` : tous les autres.
- Exemple (`ls -l`) :  
`-rw-r--r-- 1 moy moy 34004 mars 20 08:59 cm-admin.tex`

# Gestion des droits unix

- Les droits sont les droits sur les fichiers (tout est fichier).
- Les droits de base sont :
  - ▶ read lecture du fichier, liste du contenu du répertoire ;
  - ▶ write écriture dans le fichier, ajout/suppression de fichier dans le répertoire ;
  - ▶ execute exécution du fichier, aller dans le répertoire *ou un sous répertoire*.
- Pour un utilisateur, un fichier est dans l'une des classes :
  - ▶ user, u : propriétaire ;
  - ▶ group, g : groupe du propriétaire ;
  - ▶ other, o : tous les autres.
- Exemple (`ls -l`) :  
`-rw-r--r-- 1 moy moy 34004 mars 20 08:59 cm-admin.tex`
- Modifier les droits :
  - ▶ `chmod go-rx` = supprimer les droits x et r à group et other.
  - ▶ `chmod a+x` = donner le droit x à tout le monde (all)
  - ▶ `chmod 644` = donner les droits `rw-r--r--` (notation octale,  $r = 4$ ,  $w = 2$ ,  $x = 1 \Rightarrow rw- = 4 + 2 = 6$ )
- root a toujours le droit de tout faire !

# Exemple

- Pour mettre en place sa page internet personnelle, il faut que l'utilisateur *apache* ou *html* ait le droit de lire le contenu du répertoire `~/public_html/` donc :
  - ▶ `~/` doit être autorisé en exécution pour les autres.
  - ▶ `~/public_html/` doit être autorisé en exécution pour les autres.
  - ▶ `~/public_html/*.html` doivent être autorisés en lecture pour les autres.
  - ▶ `chmod go-r ~/public_html/` : sécurité faible (en l'absence de meilleure solution)
- Les mots de passe doivent être protégés . Mais la commande `passwd` doit permettre de lire et modifier le mot de passe.
  - ▶ `/etc/shadow` est en lecture uniquement pour son propriétaire `root`
  - ▶ `/usr/bin/passwd` appartient à `root`, est autorisé en exécution pour tous avec un `setUID` bit = 1 (le processus appartiendra à `root`)
- Les droits permettent de protéger le système tout en déléguant des droits étendus via certaines commandes.



# ACL

Le système de droits n'est pas suffisant :

- Il n'y a pas de droits négatifs (tous sauf ...).
  - ▶ Inversement, par exemple avec apache, les accès aux URL reposent sur allow et deny et un ordre de lecture des droits
- Seulement 3 types de personnes
  - ▶ Fastidieux, pour gérer finement les droits, les utilisateurs doivent être dans de nombreux groupes
  - ▶ Quand un utilisateur crée un fichier, à quel groupe appartient-il ?
- Une solution est d'associer à chaque objet une liste de droits (ou déni de droits) accordés à des utilisateurs ou des groupes. Ce sont les *Access Control List* ou *ACL*.



# ACL (suite)

- Une ACL est une liste d'*ACE* (*E*ntry)
- Les droits sont positifs ou négatifs
- Un ACE est formé :
  - ▶ d'un droit particulier (lecture, écriture, contrôle total, changer les droits...);
  - ▶ d'un utilisateur ou d'un groupe;
  - ▶ d'un objet sujet;
  - ▶ d'un booléen Allow ou Deny.
- On doit définir un ordre de lecture
- Exemple
  - ▶ Windows (droits de base, droit sur NTFS), OS X, Linux (setfacl, getfacl, peu utilisés en pratique)
  - ▶ ldap, firewall, AFS
  - ▶ Forums, blogs ...





- 1 Les utilisateurs
  - Gestion des utilisateurs
  - Gestion des droits
- 2 Services, systemd
- 3 Outils de diagnostic
- 4 Installation
  - Première installation
  - Installation de logiciel
  - Gestion des paquets

# Lancement d'un programme

Un programme peut être lancé de plusieurs manières :

- ① Par un utilisateur (en ligne de commande, graphiquement, ...)
- ② Au démarrage de la machine (exemple : serveur web)
- ③ Suite à un évènement (connexion réseau sur un port, branchement d'un périphérique, appui sur le bouton on/off du PC, ...)



# Lancement d'un programme

Un programme peut être lancé de plusieurs manières :

- 1 Par un utilisateur (en ligne de commande, graphiquement, ...)  
~→ vous connaissez : shell (graphique ou non)
- 2 Au démarrage de la machine (exemple : serveur web)
- 3 Suite à un évènement (connexion réseau sur un port, branchement d'un périphérique, appui sur le bouton on/off du PC, ...)

# Lancement d'un programme

Un programme peut être lancé de plusieurs manières :

- ① Par un utilisateur (en ligne de commande, graphiquement, ...)  
~> vous connaissez : shell (graphique ou non)
- ② Au démarrage de la machine (exemple : serveur web)  
~> il faut écrire quelque part quels logiciels sont lancés
- ③ Suite à un évènement (connexion réseau sur un port, branchement d'un périphérique, appui sur le bouton on/off du PC, ...)  
~> il faut écrire quelque part quelles actions doivent être réalisées suite à quel évènement

# Lancement d'un programme

Un programme peut être lancé de plusieurs manières :

- ❶ Par un utilisateur (en ligne de commande, graphiquement, ...)  
~> vous connaissez : shell (graphique ou non)
- ❷ Au démarrage de la machine (exemple : serveur web)  
~> il faut écrire quelque part quels logiciels sont lancés
- ❸ Suite à un évènement (connexion réseau sur un port, branchement d'un périphérique, appui sur le bouton on/off du PC, ...)  
~> il faut écrire quelque part quelles actions doivent être réalisées suite à quel évènement

Pour les cas 2 et 3 :

- Historiquement : un outil par type d'évènement (SysVinit au démarrage, inetd pour réagir à une connexion réseau, udevd pour gérer les périphériques, ...)
- Tendance actuelle : « Un outil pour les gouverner tous ». systemd adopté par la majorité des distributions (RedHat, Debian, Ubuntu, ...).



# systemd : commandes et configuration

- Fichiers de configuration :

- ▶ `/etc/systemd/system/*/nom.service` : descriptions des services.
- ▶ `/etc/systemd/system/*/nom.target` : descriptions d'ensembles de services (par exemple `multi-user.target`, `graphical.target`)

- Commandes (root nécessaire) :

- ▶ `systemctl start nom.service`
- ▶ `systemctl stop nom.service`
- ▶ `systemctl status nom.service`

- Exemple : serveur web (nginx) :

```
$ cat /lib/systemd/system/nginx.service
```

```
[Unit]
```

```
Description=A high performance web server and a reverse proxy server
```

```
After=network.target
```

```
[Service]
```

```
ExecStart=/usr/sbin/nginx -g 'daemon on; master_process on;'
```

```
ExecStop=-/sbin/start-stop-daemon --quiet --stop --retry QUIT/5 --pidfile /run/nginx.pid
```



- 1 Les utilisateurs
  - Gestion des utilisateurs
  - Gestion des droits
- 2 Services, systemd
- 3 Outils de diagnostic
- 4 Installation
  - Première installation
  - Installation de logiciel
  - Gestion des paquets

# Résolution de problème

« *Tout programme non trivial possède au moins un bug.* »

Corollaire de la loi de Murphy.

- Il est donc nécessaire de savoir trouver et corriger les problèmes.





# Résolution de problème

« *Tout programme non trivial possède au moins un bug.* »

Corollaire de la loi de Murphy.

- Il est donc nécessaire de savoir trouver et corriger les problèmes.
- Les systèmes donnent beaucoup d'informations qui généralement permettent de trouver la solution.



# Résolution de problème

« *Tout programme non trivial possède au moins un bug.* »

Corollaire de la loi de Murphy.

- Il est donc nécessaire de savoir trouver et corriger les problèmes.
- Les systèmes donnent beaucoup d'informations qui généralement permettent de trouver la solution.
- Mais il faut savoir où chercher :



# Résolution de problème

« *Tout programme non trivial possède au moins un bug.* »

Corollaire de la loi de Murphy.

- Il est donc nécessaire de savoir trouver et corriger les problèmes.
- Les systèmes donnent beaucoup d'informations qui généralement permettent de trouver la solution.
- Mais il faut savoir où chercher :
  - ▶ *Historique des événements (logs)* du système.



# Résolution de problème

« *Tout programme non trivial possède au moins un bug.* »

Corollaire de la loi de Murphy.

- Il est donc nécessaire de savoir trouver et corriger les problèmes.
- Les systèmes donnent beaucoup d'informations qui généralement permettent de trouver la solution.
- Mais il faut savoir où chercher :
  - ▶ *Historique des événements (logs)* du système.
  - ▶ Service en mode *debug*.



# Résolution de problème

« *Tout programme non trivial possède au moins un bug.* »

Corollaire de la loi de Murphy.

- Il est donc nécessaire de savoir trouver et corriger les problèmes.
- Les systèmes donnent beaucoup d'informations qui généralement permettent de trouver la solution.
- Mais il faut savoir où chercher :
  - ▶ *Historique des événements (logs)* du système.
  - ▶ Service en mode *debug*.
  - ▶ Utilitaires.



# Historique des événements

- Comme tous programmes, les services systèmes rendent compte de leurs actions.
- Ces messages sont centralisés et rassemblés `/var/log/`.
  - ▶ `/var/log/message` pour la plupart des logs.
  - ▶ `/var/log/httpd/*` ou `/var/log/apache2/*` pour le serveur web
  - ▶ `Xorg.0.log` pour le serveur graphique
  - ▶ ...
- Il est souvent très instructif de suivre les logs système pour voir en temps réel les effets d'une action.  
`tail -f /var/log/apache2/error.log`
- Tous les services ont dans leur configuration un *niveau de log* qui permet d'augmenter le nombre d'informations disponibles.

# Message d'erreur

- Généralement, les services sont lancés en tâche de fond, dans un mode complexe (multithread/multiprocessus,...).
- Mais ils peuvent être lancés en avant-plan pour la correction de problèmes.
  - ▶ `dhcpcd -f` (attention, différent de `dhcpcd !`)
  - ▶ `httpd -X` (<https://httpd.apache.org/docs/2.4/programs/httpd.html>)
  - ▶ `slapd -d 3`
  - ▶ ...
- Cela permet de les lancer dans un debugger, ou d'obtenir tous les messages d'erreur de l'application.

# Outils

- Debugger gdb, ddd, kdbg.





# Outils

- Debugger gdb, ddd, kdbg.
- Pour un script shell, utiliser `sh -x` qui affiche les commandes avant de les exécuter.

# Outils

- Debugger gdb, ddd, kdbg.
- Pour un script shell, utiliser `sh -x` qui affiche les commandes avant de les exécuter.
- Utilitaires d'écoute sur le réseau : `tcpdump`, `wireshark`.



# Outils

- Debugger gdb, ddd, kdbg.
- Pour un script shell, utiliser `sh -x` qui affiche les commandes avant de les exécuter.
- Utilitaires d'écoute sur le réseau : `tcpdump`, `wireshark`.
- `strace` qui affiche les appels système d'un programme.



# Outils

- Debugger gdb, ddd, kdbg.
- Pour un script shell, utiliser `sh -x` qui affiche les commandes avant de les exécuter.
- Utilitaires d'écoute sur le réseau : `tcpdump`, `wireshark`.
- `strace` qui affiche les appels système d'un programme.
- `ltrace` qui affiche les appels à une bibliothèque et leur paramètres.



# Outils

- Debugger gdb, ddd, kdbg.
- Pour un script shell, utiliser `sh -x` qui affiche les commandes avant de les exécuter.
- Utilitaires d'écoute sur le réseau : `tcpdump`, `wireshark`.
- `strace` qui affiche les appels système d'un programme.
- `ltrace` qui affiche les appels à une bibliothèque et leur paramètres.

## Règles élémentaires

- Stopper les services de cache `nsd`, de sécurité `firewall`, `selinux`.
- Rechercher les options de sécurité par défaut.

- 1 Les utilisateurs
  - Gestion des utilisateurs
  - Gestion des droits
- 2 Services, systemd
- 3 Outils de diagnostic
- 4 Installation
  - Première installation
  - Installation de logiciel
  - Gestion des paquets

# Installation

- Installer un logiciel signifie
  - ▶ Copier les exécutable
  - ▶ Copier les bibliothèques internes au logiciel
  - ▶ Installer les dépendances
  - ▶ Adapter les configurations
  - ▶ Se souvenir des modifications
- On distingue 2 types d'installations avec des utilitaires différents
  - ▶ Installation du système et des logiciels de base
  - ▶ Installation d'un logiciel particulier

# Bibliothèque (*librairies*)

- Une bibliothèque est un ensemble de codes de fonctions.
- Ces fonctions ne sont pas écrites dans chaque exécutable ni copiées dans la mémoire de chaque processus.
- Mais partagées entre processus.
- Cela signifie qu'un logiciel qui utilise ces bibliothèques dépend de fichier(s) externe(s).



# Bibliothèque (*libraries*)

- Une bibliothèque est un ensemble de codes de fonctions.
- Ces fonctions ne sont pas écrites dans chaque exécutable ni copiées dans la mémoire de chaque processus.
- Mais partagées entre processus.
- Cela signifie qu'un logiciel qui utilise ces bibliothèques dépend de fichier(s) externe(s).
- La gestion des dépendances pose problème :
  - ▶ développement des applications :
    - ★ gestion des fichiers d'entête .h : `#include`, `-I`;
    - ★ gestion des fichiers de code : `-L` ..., `-lpthread`, `-lm`
  - ▶ installation (dépendance)
  - ▶ mise-à-jour



# Exemple avec les bibliothèques

- Exemple :
  - ▶ libX11 permet l'accès à l'interface graphique
  - ▶ libgtk fournit des constructions de plus haut niveau par dessus libX11 (libgtk dépend de libX11)
  - ▶ emacs et geany sont des exécutables qui utilisent libgtk (ils dépendent de libgtk donc de libX11), xclock dépend de libX11.
- Donc ...
  - ▶ Installer emacs demande d'installer libX11 et libgtk
  - ▶ Mais on n'installe les bibliothèques qu'en un seul exemplaire (même si on a emacs et geany installés)

- 1 Les utilisateurs
  - Gestion des utilisateurs
  - Gestion des droits
- 2 Services, systemd
- 3 Outils de diagnostic
- 4 **Installation**
  - **Première installation**
  - Installation de logiciel
  - Gestion des paquets

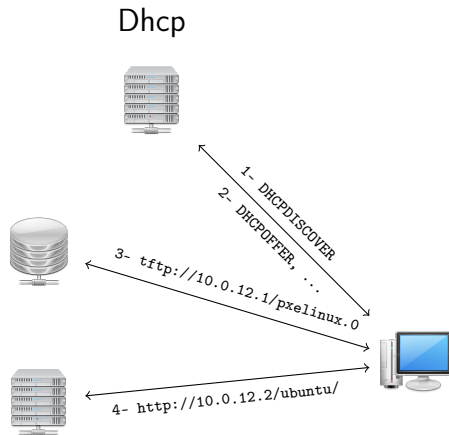
# Installation d'un système

On peut utiliser

- Un CD
  - ▶ système préconfigurés (windows, mac OS)
  - ▶ système simple à installer (linux)
- Un système de copie d'image disque (ex : ghost)
  - ▶ très efficace ;
  - ▶ mais très spécifique ;
  - ▶ de retour avec la virtualisation.
- Installation automatique
  - ▶ RedHat, Fedora : kickstart, Debian : preseeding
  - ▶ Windows : WDS Windows Deployment Service.



# Principe de l'installation automatique



- Démarrage par PXE ;
  - ▶ Récupération d'une adresse automatique (DHCP)
  - ▶ Récupération d'une mini image d'installation (TFTP)
- Installation automatisée ;
  - ▶ Récupération des paquets systèmes (HTTP, SMB, ...)
  - ▶ Récupération d'un script d'installation

# Installation automatique

	Partage de fichiers	configuration	noyau d'installation	système
Kickstart /pre-seeding	nfs, http, ftp	fichier	linux sur CD ou PXE	linux
WDS	smb	fichier	WindowsPE PXE	windows

- Avantages :
  - ▶ C'est une installation donc s'adapte au matériel
  - ▶ Permet différents scénarios
  - ▶ Automatique (pas d'intervention)
- Inconvénients :
  - ▶ Long
  - ▶ Uniquement le système de base
  - ▶ Ne permet pas facilement la configuration



# Provisioning

Il existe des logiciels facilitant ces différentes configurations :

- configuration automatique des différents serveurs (DHCP, TFTP, HTTP...);
- gestion des images système disponibles ;
- gestion de modèles de provisionnement ;
- choix à distance entre installation, démarrage normal, récupération.

Par exemple Foreman, intercafe, WDS.

- Ils permettent la gestion de parc de serveurs ou de cluster.
- Ils proposent une gestion indifférenciée de machines physiques (*bare metal*) ou virtuelles.
- Ils sont associés à des outils de supervision, orchestration.



- 1 Les utilisateurs
  - Gestion des utilisateurs
  - Gestion des droits
- 2 Services, systemd
- 3 Outils de diagnostic
- 4 **Installation**
  - Première installation
  - **Installation de logiciel**
  - Gestion des paquets



# Quel est la différence

Pourquoi faire la différence entre installation de logiciel et de la machine ?

- Installation sur des systèmes qui évoluent
- Les scripts d'installation automatique existent mais il n'y a pas de standard
- Installation plus simple
  - ▶ copie de fichiers
  - ▶ peu de configuration
- Entretien plus complexe
  - ▶ Mise-à-jour
  - ▶ Suppression
- L'éditeur du logiciel fournit un programme d'installation.
- Pour régler les problèmes, on utilise un système de paquets



# Les paquets

1 paquet = 1 « archive » contenant :

- Les fichiers à copier sur le système.
- Les configurations.
- Un script d'installation et de désinstallation.
- Les dépendances (possibilité).
- Avantages :
  - ▶ Le gestionnaire de paquets se souvient des installations
  - ▶ C'est automatisable
  - ▶ Dépendances
- Inconvénients :
  - ▶ L'éditeur ne fournit pas toujours un paquet
  - ▶ Surtout les logiciels avec leur propre système de paquets (emacs, python, perl, matlab, R...)
  - ▶ Dépendances

Cas particuliers : Gentoo, Funtoo, ArchLinux



# Exemple : les paquets .rpm (Fedora/RedHat/Suse/Mandrake) et .deb (Debian/Ubuntu)

- Les sources du logiciel et patch
- Des scripts
  - ▶ Compilation
  - ▶ Pre/post installation
  - ▶ Pre/post déinstallation
- Liste des fichiers installés.
- 2 types de paquets, source et binaire.
- On crée un paquet en compilant le logiciel.
- Les dépendances sont données à la main ou calculées automatiquement (moins clairement).
- Avantages/Inconvénients
  - + fiabilité
  - + portabilité
  - demande beaucoup de connaissance sur le logiciel à installer.

<https://www.debian.org/doc/manuals/debian-faq/pkg-basics.fr.html>



- 1 Les utilisateurs
  - Gestion des utilisateurs
  - Gestion des droits
- 2 Services, systemd
- 3 Outils de diagnostic
- 4 **Installation**
  - Première installation
  - Installation de logiciel
  - **Gestion des paquets**

# Gestionnaires

L'intérêt des paquets est d'automatiser l'installation.

- Utilisation de gestionnaires capables
  - ▶ D'aller chercher les paquets en local ou sur internet (*dépôts* ou *repository*) de façon *sécurisée*
  - ▶ De gérer des groupes de logiciel.
  - ▶ De gérer les mises-à-jour.
  - ▶ De gérer les dépendances.
- Permet de déployer des logiciels
- Permet de gérer des configurations logicielles

# Lesquels 1/2

## Linux

- rpm/dnf (Fedora, RedHat), dpkg/apt/aptitude/dselect (Debian, Ubuntu), emerge (Gentoo, Funtoo), pacman (Archlinux)
- + mise-à-jour et installation
- + dépôts fiables
- Pas vraiment adapté à la gestion de configurations logiciels

<https://www.debian.org/doc/manuals/debian-faq/pkgtools.fr.html>

<https://www.debian.org/doc/manuals/debian-faq/uptodate.fr.html>

## Remarques

- `/etc/apt/sources.list`, `/etc/apt/preferences`, `/var/cache/apt/archives/`
- Des gestionnaires de paquets *dans* les applications
  - ▶ MEPLA pour emacs
  - ▶ les gems pour Ruby : `gem -help`
  - ▶ les python packages, avec `pip`

# Lesquels 2/2

## Windows

- Windows update (mise-à-jour) Stratégie de groupe AD (installation)
  - pas de dépôts
  - mise-à-jour
- + gestion très fine des logiciels

# Sous Debian GNU/Linux : dpkg/apt

- Debian PacKaGe (dpkg) :
  - ▶ `dpkg --install toto.deb` : installe le paquet `toto.deb` (présent sur le disque)
  - ▶ `dpkg -l` : liste les paquets déjà installés
  - ▶ `dpkg --getfiles toto` : liste les fichiers installés par le paquet `toto` (déjà installé)
- Advanced Package Tool (apt), surcouche à dpkg :
  - ▶ `apt update` : télécharge la liste des paquets disponibles depuis les dépôts
  - ▶ `apt search` : cherche dans la liste des paquets disponibles
  - ▶ `apt policy` : voir les versions disponibles/installées d'un paquet
  - ▶ `apt install` : télécharge puis installe un paquet
  - ▶ `apt upgrade` : met à jour tous les paquets installés (à faire régulièrement pour des questions de sécurité, penser au `apt update` avant)
- Interfaces graphiques :
  - ▶ `aptitude` (ncurses)
  - ▶ `synaptic` (graphique)
  - ▶ `update-manager` (graphique, Ubuntu, juste pour les mises à jour)





# Sous Debian GNU/Linux : dpkg/apt

À lire avant le TP :

- <https://www.debian.org/doc/manuals/debian-faq/pkg-basics.fr.html>
- <https://www.debian.org/doc/manuals/debian-faq/pkgtools.fr.html>
- <https://www.debian.org/doc/manuals/debian-faq/uptodate.fr.html>

# Automatiser l'automatisation ...

Déployer sur N machines :

- « à l'ancienne » :

```
for m in $(cat machines.txt); do
    ssh $m apt install $package
done
```

- Utiliser des outils standards :

- ▶ Ansible :

```
ansible all -s -m apt -a \
    'pkg=nginx state=installed update_cache=true'
```

- ▶ Puppet :

```
vi manifest.pp
puppet apply manifest.pp
```

- ▶ Chef :

```
sudo chef-apply -e "package 'emacs'"
```



# Conclusion

- Installation

- ▶ Recherche dans les dépôts
- ▶ Recherche de paquets
- ▶ Création d'un paquet
- ▶ Installation à la main

- Administration

- ▶ Il faut comprendre de qu'on fait.
- ▶ Il faut être capable de l'adapter.