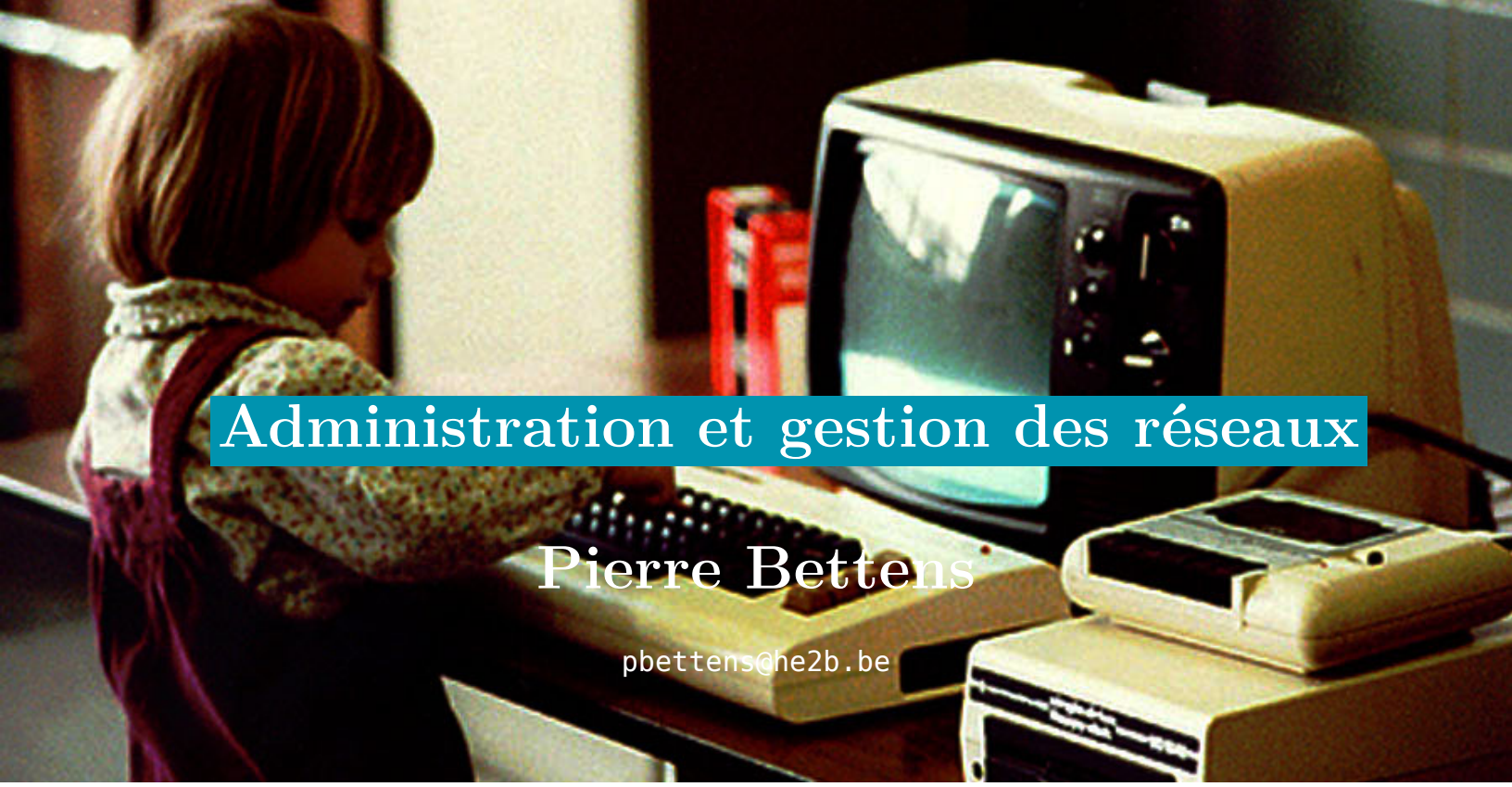


Administration et gestion des réseaux

Pierre Bettens

pbettens@he2b.be

février 2021

HE2B ESI

root est-il Dieu ? 3

*nix, les bases 4

Rappel des bases réseau 25

Le shell 32

Monitoring et fichiers de logs 42

DNS domain name system 43

Todos 54

Bibliographie 56

Administration et gestion des réseaux

version pre pre alpha

Date de publication 11 mars 2021

Remercions les auteurs des excellents livres TCP/IP de Craig Hunt[1], Administration réseau sous Linux de Olaf Kirch et Terry Dawson[2] qui ont servi de référence pour ces notes ainsi que les rédacteurs, rédactrices, traducteurs et traductrices des pages de manuel et de Wikipedia.

FIXME

J'en profite pour remercier tout particulièrement toutes celles et tous ceux qui aiment bien être remerciés.

[1] *TCP/IP Administration des réseaux*

[2] *Administration réseaux sous Linux*

[3] *Licence creative Common BY-NC-SA 4.0* http://creativecommons.org/licences/by-nc-sa/4.0/

root est-il Dieu ?

> Un grand pouvoir implique de grandes responsabilités.
>
> *Benjamin Parker*

Les tâches d'administration sont variées, elles demandent des compétences techniques et une certaine réserve.

Dans la suite, nous appellerons la personne avec le rôle d'administration : *root*.

Les tâches techniques de *root* gagnent souvent à être centralisées, sont souvent critiques au niveau de la confidentialité et de la sécurité. C'est tâches peuvent être du support de première ligne, de l'installation de machines et de services, de la maintenance et la mise à jour desdits services, la gestion des sauvegardes (*backups*), la réalisation et le maintien de la documentation, etc.

Il n'est pas nécessaire d'être un développeur ou une développeuse pour prendre en charge l'administration des réseaux mais il est essentiel de bien connaitre *son* éditeur de texte ou de code[4] et son interprèteur de commandes. Être à l'aise avec l'écriture de scripts `bash` ou autre.

Les actions faites par *root* ont souvent un impact sur les utilisateurices[5]. Elles peuvent les empêcher de travailler. Une bonne résistance au stress et une capacité à être *multi-tâche* aidera à répondre aux demandes. Il sera parfois nécessaire de dire non à des demandes qui ne sont pas réalisables — bien qu'elles puissent paraitre[6] naturelles à la personne demandeuse qui n'a pas du vue globale sur le système d'information.

Dans son rôle d'administration, *root* a accès à des informations confidentielles. Iel doit avoir une éthique irréprochable.

Mais qui est *root* ?

Sous les systèmes **nix*, *root* est l'utilisateur privilégié, le « super utilisateur », l'administrateur… C'est le compte ayant le *userid* `0`.

[4]Un *éditeur de texte* est un programme qui permet d'écrire du texte (sans mise en forme) et de le sauver. Un *éditeur de code* est un programme qui permet d'écrire du texte et qui offre des services à l'utilisateurice ; coloration syntaxique, indentation, raccourcis clavier… Nous ne parlons bien sûr pas de *traitement de texte*. `notepad` est un éditeur de texte, `vim`, `notepad++` sont des éditeurs de code, `LibreOffice Writer` est un traitement de texte. Ceci étant dit, mon conseil est d'être à l'aise avec `vim` et d'oublier `nano`.

[5]Dans ces notes, l'écriture tente d'être inclusive. Un peu pour céder à la mode, un peu parce que ça « m'amuse » et un peu pour essayer d'inclure plus. J'essaierai de ne pas en abuser. Je troque généralement l'accord dit « du masculin l'emporte » contre l'accord de proximité et j'ajoute quelques nouveaux mots en évitant le point médian · qui fait peur ;-)

[6]Ces notes sont écrites en orthographe réformée.

*nix, les bases

Les utilisateurices *users* et les groupes **4**
Choisir un bon mot de passe 7
Le coin des commandes 8
Le système de fichiers **10**
Chemins relatifs, chemins absolus 12
Les différents types de fichiers 12
Les permissions 13
Le coin des commandes 13
Les processus **16**
Le coin des commandes 17
Tâches périodiques `cron` **19**
Gestionnaire de paquet `apt` **21**
`apt` 21
Démarrage du système, `systemd` **23**
Le coin de la commande 24

Cette section rappelle les bases des systèmes **nix*. Nous supposons dans la suite que la personne lisant ces notes a déjà quelques connaissances des systèmes **nix*. Nous ne rappelons ici que ce qui nous semble nécessaire à l'administration **nix* dans le cadre de ce cours.

Nous nous concentrons sur *linux*.

Les utilisateurices *users* et les groupes

> Un système correctement administré a plus de *groups* que de *users*.

Lors de l'installation, un compte administrateur ayant tous les privilèges est créé. Le compte *root*. En plus de ce compte au minimum un compte sans privilège particulier est ajouté. Nous l'appellerons *user* même si le nom peut être choisi.

La (première) bonne pratique est de se connecter au système avec un compte non privilégié — le compte *user* — et de n'utiliser les privilèges de *root* que lorsque c'est nécessaire.

root devra gérer les utilisateurs du système dans deux situation différentes :

- la première si le système doit être accessibles par plusieurs utilisateurices, il sera nécessaires de gérer les comptes.

C'est évident ;

- la seconde est pour l'installation de services sur le système. Une bonne pratique est d'associer un compte à chaque service afin que chaque service tourne avec les privilèges de ce compte… et pas ceux de *root*.

  Par exemple, le service *dns* tourne sous le compte *bind*, le serveur web *apache2* avec le compte *www-data*…

Créer un compte consiste en l'enregistrer dans le système en lui donnant un *login*, *password*, un répertoire *home*, un *shell*… tout cela se traduit par une ligne dans le fichier `/etc/passwd` et une autre dans `/etc/shadow`.

```
$ cat /etc/passwd
login:passwd:uid:gid:comment:home:shell
user:x:1000:1000:user,,,:
  /home/user:/bin/bash
```

- `login` le nom associé au compte. La bonne pratique est d'utiliser un *login* en minuscules, sans accents ni caractères spéciaux. La longueur est limitée à 16 caractères;

- `passwd` contenait anciennement le mot de passe hashé associé au compte. Aujourd'hui, *linux* ne laisse plus un hash en lecture et le mot de passe ne se trouve plus[7] dans `/etc/passwd`.

  Le champ du mot de passe chiffré peut être vide. Dans ce cas, aucun mot de passe n'est nécessaire pour s'authentifier avec le compte donné. Cependant, certaines applications qui lisent le fichier `/etc/passwd` peuvent décider de ne donner aucun accès si le mot de passe est vide. Si le mot de passe est un « `x` » minuscule, alors le mot de passe chiffré se trouve dans le fichier *shadow* (`man 5 shadow`) ; il doit y avoir une ligne correspondante dans le fichier shadow, sinon le compte de l'utilisateur n'est pas valide. Si le mot de passe est constitué d'une autre chaîne, alors il est considéré comme un mot de passe chiffré, comme indiqué dans *crypt* (`man 3 crypt`).

  Ce champ peut aussi prendre la valeur « * » qui précise qu'il n'est pas possible de se connecter au compte avec `login` ou la valeur « `!` » qui empêche toute connexion au compte. Ces valeurs sont généralement utilisées pour les comptes de service. Les comptes qui sont utilisés par les services et qui ne sont pas destinés pour qu'une personne puisse s'y logger.

- `uid` est l'*user id*, l'identifiant unique de l'utilisateur. La valeur 0 est celle de *root*, les autres sont libres. Les premières valeurs sont utilisées par le système. En fonction des distributions les *uid* des comptes commencent à partir d'une certaine valeur. Pour *debian*, c'est 1000.

  Attention certaines applications se basent sur l'*uid* et pas le *login* du compte. Il s'agit donc d'agir avec prudence lorsque des machines communiquent entre elles pour ces quelques services (par exemple `nfs`) ;

- `gid` est le *group id*, l'identifiant unique du groupe. Chaque compte est associé à un groupe principal et peut être ajouté à d'autres groupes. Le groupe

[7] S'il s'y trouve il faut revoir la sécurité de cette machine.

principal est renseigné ici, les autres dans `/etc/group`[8] ;

- `comment` série de valeurs reprenant le nom complet du compte ainsi que diverses informations ;
- `home` répertoire *home* du compte. Chemin absolu ;
- `shell` shell associé au compte. Généralement `/bin/bash`.

  Cette valeur peut être positionnée à `/bin/false` pour désactiver le compte ou pour empêche un login au compte.

Le fichier `/etc/shadow` est un fichier qui contient les informations cachées concernant les mots de passe des comptes et leurs dates de validité.

```
# cat /etc/shadow
user:$6$0--cut--Z3U/:17309:0:99999:7:::
```

Ce fichier ne doit pas être accessible en lecture par les utilisateurices normaux afin de maintenir la sécurité des mots de passe, en particuliers pour prévenir les attaques par dictionnaires.

Chaque ligne de ce fichier contient 9 champs, séparés par des deux-points (« `:` »), dans l'ordre suivant :

- `login` le login — existant — du compte concerné ;
- `password` le mot de passe *hashé*.

  Si le champ du mot de passe contient une chaine qui ne peut pas être un résultat valable de crypt(`man 3 crypt`), par exemple si elle contient les caractères `!` ou `*`, alors l'utilisateur ou l'utilisatrice ne pourra pas utiliser son mot de passe UNIX pour se connecter (mais il se peut que le compte puisse se connecter au système par d'autres moyens).
- `date du dernier changement de mot de passe` la date du dernier changement de mot de passe, exprimée en nombre de jours depuis le 1er janvier 1970.

  Quand cette valeur vaut `0` un changement de mot de passe est requis à la prochaine connexion.

  Quand la valeur est absente (champ vide), les fonctionnalités de vieillissement de mot de passe sont désactivées.
- `âge minimum du mot de passe` l'âge minimum du mot de passe est la durée (en jour) que l'utilisateur devra attendre avant de pouvoir le changer de nouveau.

  Un champ vide ou une valeur de 0 signifie qu'il n'y a pas d'âge minimum pour le mot de passe.
- `âge maximum du mot de passe` l'âge maximum du mot de passe est la durée (en jour) après laquelle l'utilisateur devra changer son mot de passe.

  Une fois cette durée écoulée, le mot de passe restera valable. Il sera demandé à l'utilisateur de le changer la prochaine fois qu'il se connectera. Un champ vide signifie qu'il n'y a pour le mot de passe aucune limite d'âge, aucune période d'avertissement d'expiration et aucune période d'inactivité (voir ci-dessous).
- `période d'avertissement d'expi-`

[8] Les lignes sont de la forme `group:password:gid:users_list`. La commande `groups` donne la liste des groupe d'un *user*. Un *grep* pourrait faire l'affaire `grep user /etc/group`.

`ration du mot de passe` la durée (en jour) pendant laquelle l'utilisateur sera averti avant que le mot de passe n'expire (voir l'âge maximum du mot de passe ci-dessus).

Un champ vide ou une valeur de 0 signifie qu'il n'y aura pas de période d'avertissement d'expiration du mot de passe.

- `période d'inactivité du mot de passe` la durée (en jour) pendant laquelle le mot de passe sera quand même accepté après son expiration (voir l'âge maximum du mot de passe ci-dessus). L'utilisateur devra mettre à jour son mot de passe à la prochaine connexion.

  Après expiration du mot de passe suivie de la période d'expiration, plus aucune connexion n'est possible en utilisant le mot de passe de l'utilisateur. L'utilisateur doit contacter son administrateur. Un champ vide signifie qu'aucune période d'inactivité n'est imposée. date de fin de validité du compte La date d'expiration du compte, exprimé en nombre de jours depuis le 1er janvier 1970.

  Un champ vide signifie que le compte n'expirera jamais.

  La valeur 0 ne doit pas être utilisée puisqu'elle peut être interprétée soit comme un compte sans expiration, soit comme ayant expiré le 1er janvier 1970.

- le dernier champ est réservé pour une utilisation future.

Outre ces deux entrées dans les fichiers `/etc/passwd` et `/etc/shadow` la création d'un compte entraine la copie des fichiers contenu dans `/etc/skel` dans le répertoire *home* du compte. C'est là que *root* peut paramétrer certains fichiers de configuration avant la création d'un compte. Par défaut, ce répertoire contient :

```
alice@harmony:~$ tree -a /etc/skel
/etc/skel
├── .bash_logout
├── .bashrc
└── .profile

0 directories, 3 files
```

Un manière simple de désactiver un compte est de changer son *shell* dans le fichier `/etc/passwd` et le remplacer par `/bin/false` par exemple.

Choisir un bon mot de passe

Le choix d'un bon mot de passe est primordial pour les mots de passe sensibles comme les mots de passe des comptes administrateurs et donc, *root* mais également pour les mots de passe des comptes utilisateurs sans privilège particulier. À ce sujet, il sera possible de mettre en place une *politique de mot de passe*.

En préambule, respecter les règles de bonne utilisation des mots de passe est contraignant et difficile. C'est cependant un des éléments principaux de la sécurité informatique.

Voici quelques règles habituelles :

- un mot de passe devrait être spécifique à un service informatique et ne devrait pas être réutilisé~: un service = un mot de passe spécifique et unique;
- un mot de passe est personnel et ne peut être donné à personne;
- un bon mot de passe est facile à retenir pour l'utilisateur qui le définit et dif-

ficile à trouver par une machine ou quelqu'un d'autre;

- un mot de passe ne devrait pas être basé sur des informations personnelle qui peuvent être facilement identifiées ou devinées;
- une bonne manière de faire est qu'il soit long, et ne puisse pas figurer dans une liste ou un dictionnaire;
- pour définir un bon mot de passe, par exemple accoler des mots en incorporant des majuscules, des minuscules, des chiffres et des caractères spéciaux (car un mot de passe ne peut souvent pas contenir d'espace). Ou encore prendre les initiales des mots d'une phrase. C'est mieux également s'il mélange les langues;

  Par exemple si je pense aux trois mots « table », « chaise » et « manger », un bon mot de passe pourrait être `tableCHAISEeat723`.

  De part sa longueur ce mot de passe sera résistant — à l'heure où j'écris — à une force brute caractère par caractère mais aussi à une recherche mot par mot puisqu'il utilise deux langues et des chiffres.

- pour un mot de passe de service, il est inutile qu'il soit retenu et il est tout à fait envisageable de le stocker dans un gestionnaire de mots de passe.

Nous verrons avec `PAM` qui est possible de définir des règles sur le format des mots de passe.

Le coin des commandes

`adduser` et `addgroup`

`adduser` et `addgroup` ajoutent des comptes utilisateurs ou des groupes au système en fonction des options et du fichier de configuration `/etc/adduser.conf`. Ces commandes choisissent un *uid* et *gid* conformes à la charte *debian*, crée un répertoire personnel configuré en fonction du contenu de `/etc/skel`, d'exécuter un script sur mesure.

```
adduser <username>
```

Ajoute un compte utilisateur normal en lui associant le premier *uid* disponible et un groupe — créé si nécessaire — du même nom que le compte utilisateur.

- Attribue un masque de `002` au compte.
- Crée un répertoire personnel. Par défaut tous les fichiers créés dans le répertoire personnel du compte utilisateur auront le bon groupe.
- Copie les fichiers *skel* dans le répertoire personnel du compte utilisateur.
- Si le fichier `/usr/local/sbin/adduser.local` existe, il sera exécuté.

```
adduser --system <username>
```

Ajoute un compte utilisateur système.

- Par défaut les comptes utilisateurs systèmes sont placés dans le groupe `nogroup`.
- Un répertoire personnel est également créé.
- Le shell par défaut sera `/usr/sbin/nologin`.

```
adduser --group <groupname>
addgroup <groupname>
addgroup --system <groupname>
```

Ces trois commandes ajoutent un groupe au système sans utilisateur. La dernière ajoute un groupe système qui diffère d'un groupe normal par son *gid* qui n'est pas dans le même intervalle.

```
adduser <username> <groupname>
```

Ajoute le compte utilisateur *username* au groupe *groupname*.

useradd et usergroup

Ces commandes sont les commandes bas niveau associées aux commandes précédentes. Elles créent les compte en fonction des paramètre donnés en option sur la ligne de commande.

userdel et deluser

Comme pour les autres commandes `userdel` est la commande de bas niveau pour laquelle il est nécessaire de passer en arguments les options désirées tandis que `deluser` est de plus haut niveau.

`userdel` se contente de supprimer les entrées de `/etc/passwd` et `etc/shadow`, elle ne supprime par exemple pas le groupe principal associé au compte utilisateur. Par contre, elle ne supprimera pas le compte s'il a des processus actifs.

`deluser` et `delgroup` retirent des comptes utilisateur et des groupes du système suivant les options et les informations contenues dans `/etc/deluser.conf` et `/etc/adduser.conf`.

```
deluser <username>
deluser --remove-home \
  --remove-all-files <username>
```

Retire un compte utilisateur normal.

- Par défaut le répertoire personnel n'est pas supprimé, c'est l'option `--remove-home` qui s'en charge.
- L'option `--remove-all-files` recherche tous les fichiers que *username* possède et les supprime.
- Il est possible de faire une sauvegarde des fichiers du compte utilisateurs *via* l'option `--backup`.
- Si le fichier `/usr/local/sbin/deluser.local` existe, il sera exécuté après la suppression du compte. Ceci permet un nettoyage supplémentaire.

```
deluser --group <groupname>
delgroup <groupname>
```

Retire un groupe du système.

```
deluser <username> <groupname>
```

Retire le compte utilisateur du groupe.

chage

La commande `chage` modifie le nombre de jour entre les changements de mot de passe et la date du dernier changement.

```
chage -M 30
```

Cette commande force le changement du mot de passe chaque mois.

su

La commande `su` — pour *substitute* — permet d'exécuter une commande sous le compte d'un autre compte utilisateur.

Sans argument, `su` exécute un *shell* en *root*.

L'option `-` (ou `-l` ou `--login`) lance le *shell* comme un *shell* de login:

- efface les variables d'environnement exceptée `TERM`;
- initialises les variables d'environnement `HOME`, `SHELL`, `USER`, `LOGNAME` et `PATH`[9];
- change de répertoire vers le répertoire cible;
- place le premier argument (`argv[0]`) du shell à `-` pour en faire un shell de login.

`su` utilise PAM.

```
su -
su alice -c "ls -il"
```

Le système de fichiers

Une machine **nix* n'a qu'un seul système de fichier (*filesystem*) dont la racine se note `/`. Ce *filesystem* représente comment on accède aux « informations » stockées sur un « support ».

- *l'information* peut bien sûr être un fichier ou un programme mais elle peut aussi être un *pseudo-fichier* faisant le lien avec un composant matériel (*hardware*).
- *le support* sera une partition d'un disque dur bien sûr mais également un accès à un partage distant accessible par le réseau. Il pourra aussi être un pseudo-fichier accédant au matériel.

La structure du *filesystem* *nix est standardisée par le « Filesystem Hierarchy Standard »[10][11] — même si les différentes distributions ne respectent pas le standard à la lettre, les grandes lignes le sont.

Le groupe en charge de l'harmonisation de *filesystem* à choisi de distinguer :

- les fichiers **partageables** entre plusieurs machines de ceux qui ne le sont pas. Les pages de manuel peuvent par exemple être partagées;
- les fichiers **variables** et ceux qui le sont peu. Les fichiers variables ont un contenu et une taille qui varie fortement pendant la vie du programme qui les utilise et donc du système. Par exemple, les mails entrants et sortant.

En voici un résumé :

[9] Ce point est source d'erreurs. Une exécution de `su` — sans le `-` donc — ne charge pas le `PATH` de *root* mais conserve celui de l'utilisateurice sans privilège. Les répertoires `/sbin` et `/usr/sbin` n'en font par exemple pas partie.

[10] *Filesystem Hierarchy Standard* https://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard

[11] *Filesystem Hierarchy Standard (pdf)* https://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf

Répertoire	*Description*
`/bin`	Les commandes de base nécessaires au démarrage et à l'utilisation d'un système minimaliste (`ls`, `cat`…) exceptées les commandes *root* qui se trouvent dans `/sbin`. *bin pour binaire.*
`/boot`	Les fichiers nécessaires au démarrage (par ex. le noyau et *grub*). *boot pour… boot*
`/dev`	Les *pseudo-fichiers* correspondant à un périphérique *hardware*. *dev pour devices*
`/etc`	Fichiers de configuration. Contient souvent un répertoire pour le programme concerné (par ex. `/etc/apache2/`). *etc pour editable text configuration.*
`/home`	Répertoires des *users* du système (par ex. `/home/alice`) *home pour « qu'il fait bon chez moi ».*
`/lib`	Bibliothèques logicielles (*libraries*) nécessaires aux binaires de `/bin` et `sbin`. *lib pour libraries.*
`/mnt`	Point de montage pour les systèmes de fichiers temporaires. *mnt pour mount.*
`/media`	Point de montage pour les médias amovibles (anciennement les CD-ROM, aujourd'hui les clés USB et demain…). *media pour medias*
`/opt`	Logiciels optionnels, ce sont ceux qui ne sont pas proposés par la distribution et installés pour tous les *users*. Il est préférable d'utiliser `/opt` à `/usr/local`. *opt pour optionals.*
`/proc`	Système de fichiers virtuel pour les processus. *proc pour processes*
`/root`	Répertoire *home* de *root*.
`/sbin`	Binaires pour *root*. *sbin pour system binaries.*
`/srv`	Données pour les services hébergés (contenu web statique, base de donnée…). *srv pour services.*
`/tmp`	Fichiers temporaires. Le répertoire est vide au démarrage. *tmp pour temporary.*
`/usr`	Arborescence semblable à la racine pour les fichiers et répertoires qui ne sont pas nécessaires au fonctionnement minimals du système (par ex. `/usr/bin`, `/usr/lib`, `/usr/sbin`, `/usr/share`, `/usr/include`…) *usr pour unix system resources*
`/var`	Destiné à recevoir des fichiers variables divers. *var pour variables.*
`/var/cache`	Pour différents cache (par ex. *bind*, *apt*).
`/var/lock`	Fichiers de verrouillage.
`/var/mail`	Boites mails des utilisateurs.
`/var/spool`	Données en attentes de traitement (par ex. pour l'impression, les mails, les tâches planifiées…).

Remarques :

- Certains binaires se trouvant dans `/sbin` peuvent être exécutés par un *user* sans privilège tant que cet *user* ne demande pas une action à laquelle il ou elle n'a pas droit. Par contre `/sbin` ne se trouve pas dans son `PATH`.
- Historiquement un petit disque — rapide — était réservé aux fichiers essentiels — se trouvant dans `/bin`, `/sbin`… — tandis que les autres étaient placés dans `/usr` qui pouvait donc se trouver sur un autre disque.
- Les répertoires `/usr/src` et `/usr/include` sont plutôt destinés à recevoir tout ce qui est nécessaires à la compilation C ou C++ des différents logiciels se trouvant sur la machine.

Chemins relatifs, chemins absolus

Les noms de fichiers commençant par `/` sont des noms de fichiers **absolus** c'est-à-dire faisant référence à la racine du *filesystem*. Le nom est indépendant du répertoire courant.

Il existe un raccourci représentant le répertoire *home* de l'utilisateurice : `~`. Pour l'utilisatrice *alice*, il s'agit de `/home/alice`.

Par exemple :

```
/etc/apache2/apache2.conf
~/bin/yascript.sh
```

Les noms de fichiers **relatifs** sont relatifs au répertoire courant, ils s'expriment sans la référence à la racine `\`. Les raccourcis `.` et `..` représentent respectivement le répertoire courant et le répertoire parent.

Par exemple si le répertoire courant est `/home/alice` :

```
../../etc/apache2/apache2.conf
bin/yascript.sh
```

Les différents types de fichiers

Un fichier est désigné par un **nom** bien sûr et possède un ***inode*** unique. L'*inode* d'un fichier contient : le type de fichier, les droits d'accès, le nombre de liens physiques, un *uid* du propriétaire, un *gid*, la taille du fichier, les dates d'accès, de modifications, les connexions et l'adresse du fichier.

Les différents types de fichiers sont :

- fichier ordinaire. Quasi tout ce qui peut être enregistré est un fichier;
- répertoire (*directory*). C'est un fichier contenant une table associant un nom à un *inode*. Les noms et les *inodes* des fichiers qu'il contient;
- fichier de périphérique. Les *peusdo-fichiers* faisant le lien vers du matériel (disque, terminal, sortie parallèle, sortie série, sortie usb…). Ils se trouvent dans `/dev`;
- lien physique et lien symbolique :
  - un lien physique est un fichier

contenant l'*inode* du fichier qu'il référence. C'est un nom supplémentaire pour le même inode;

À chaque ajout d'un lien physique le nombre de référence vers le fichier augmente. À chaque suppression, il diminue. Lorsque le nombre de référence passe de 1 à 0, le fichier est supprimé.

– un lien symbolique montre le chemin vers le fichier pointé. C'est un nouveau fichier —avec un nouvel *inode*— contenant l'*inode* du fichier vers lequel le lien pointe.

```
$ echo "foo" > file
$ ln file fileln
$ ln -s file filens
$ ls -il
total 8
262155 -rw-r--r-- 2 pbt pbt 5 fév 18 22:52 file
262155 -rw-r--r-- 2 pbt pbt 5 fév 18 22:52 fileln
262158 lrwxrwxrwx 1 pbt pbt 4 fév 18 22:53 filelns -> file
```

- tube nommé (*named pipe*). Un tube nommé est un tube… nommé. Comme la commande *pipe* « `|` », un tube permet de relier la sortie d'un processus à l'entrée d'un autre. Un pipe nommé permet à un processus d'écrire dans un « _fichier sans fin ». Pendant ce temps, un autre processus peut lire dans ce « fichier sans fin ».

Les permissions

`ugo` pour *user*, *group* et *other*. Un fichier, quel que soit son type a des permission pour son propriétaire, son groupe et « les autres ». Les autres étant tous les *users* n'appartenant pas au groupe.

Pour chacun d'entre eux, les droits peuvent être `rwx` pour *read*, *write* et *execute*. Les droits d'un fichier se présentent comme suit :

```
-rwxr---wx 1 alice yagroup  46K fév 11 16:11 filename.pdf
```

- `r` permet de lire un fichier ou de voir le contenu d'un répertoire;
- `w` permet d'écrire dans un fichier ou d'écrire dans un répertoire. Écrire dans un répertoire, c'est ajouter, effacer, renommer un fichier;
- `x` indique que le fichier est exécutable (il peut s'agir d'un binaire ou d'un script) ou que le répertoire est « traversable ».

Le coin des commandes

ls

`ls` donne la liste des fichiers d'un répertoire. Voici quelques options :

- `-l`, format long donne les droits, le nombre de liens vers le fichier, le propriétaire, le groupe, la taille, la date de dernier accès et le nom;
- `-i`, affiche le numéro d'*inode* du fichier;

cd

`cd` (*change directory*) change le répertoire courant.

```
cd /home/alice
cd ..
cd ~/bin
cd -
```

L'option `-` change le répertoire vers le répertoire précédent (pas le parent).

chmod

`chmod` permet de changer le propriétaire et le groupe d'un fichier ou d'un répertoire, le *mode*. Le mode peut être représenté de manière symbolique ou octale.

- représentation symbolique `[ugoa...][-+=][perms]` où *perms* est 0 ou plus parmi les lettre `rwxXst`. Plusieurs modes symboliques peuvent être donnés, séparés par des virgules;
- représentation octale est composée de maximum 4 chiffres octaux. Le premier permet de placer le *user id* ou le *group id* ou le *sticky bit*, le second pour le *user* `u`, le troisième pour le *group* `g` et le quatrième pour *other* `o`.

  L'usage courant se compose des trois derniers.

```
chmod u+x, go-w foo
chmod 510 bar
```

chgrp

Change le groupe de chaque fichier.

```
chgrp newgroup file
```

du et df

Ces deux commandes donnent des informations sur l'espace disque. `du` (*disk usage*) donne l'espace utilisé pour un répertoire tandis que `df` (*disk free*) donne l'espace disponible sur la partition concernée.

Option `-h` donne les résultats de manière « *human readable* ».

u|mount

`mount` et `umount` permettent d'ajouter / retirer un support au système. Pour qu'un *filesystem* soit accessible il doit être monté sur un répertoire.

Les montages habituels d'un système sont ceux réalisés au *boot* du système et ceux lorsque l'utilisateurice insère une clé USB. Le premier montage est fait automatiquement grâce aux renseignements se trouvant dans le fichier `/etc/fstab` tandis que l'autre est actuellement pris en charge par un « auto-mount » dès que l'insertion de la clé est faite. En tant que *root* il sera parfois nécessaire de manipuler les différentes partitions.

```
mount
mount /dev/sdb1 /mnt/backup
```

- la première commande (sans paramètre) montre l'état des différents montages;

- le seconde monte la première partition du second disque scsi dans le répertoire `/mnt/backup` (qui doit exister au préalable).

`touch`

`touch` « touche » le fichier. L'effet est de changer la date d'accès au fichier ou de créer un fichier vide s'il n'existait pas.

`mkfifo`

`mkfifo` crée un pipe nommé (*named pipe*).

Les processus

Un processus est un programme en cours d'exécution. Un programme pouvant être exécuté plusieurs fois en même temps, il est possible d'avoir plusieurs instances du même programme au même moment.

Un processus se caractérise par :

- un *pid*, identifiant de processus (*process id*) ;
- un *ppid*, identifiant du processus parent (*parent process id*) ;
- un *uid*, l'identifiant de *user* qui a lancé le processus ;
- un *guid*, l'identifiant du groupe du *user* qui a lancé le processus ;
- un *euid*, l'identifiant de *user* « effectif » (*effective user id*) qui a lancé le processus_;
- un *egid*, l'identifiant du groupe du *user* qui a lancé le processus ;
- un *état*, état (*state*) du processus (voir ci-dessous);
- une *priorité*

gid est un alias de *egid*. *euid* est un alias de *uid*.

Un processus n'est pas toujours en cours d'éxécution (*running*) puisqu'il y a plus de processus que de CPU. La plupart des processus sont soit en cours d'exécution (***run***), soit prêt (***ready***), soit en attente (***wait***).

À sa création, un processus est placé dans le statut prêt (*ready*) et attend d'être choisi par le *scheduler*.

Un processus *running* devient « en attente » (*waiting*) lorsqu'il attend des ressources qui ne sont pas encore disponibles ; entrées-sorties ou tout autre évènement. Il se met en seul en attente ou le *kernel* s'en charge. Lorsque les ressources attendues seront disponibles, le *scheduler* mettra le processus dans l'état prêt (*ready*)._

Voici les différents états d'un processus tels que présentés dans la page de manuel :

s (*state*)	**État du processus**
`R`	*running* or *runnable*, le processus est en cours d'exécution (*running*) ou est prêt à l'être (*runnable*) (il ne lui manque que le CPU).
`S`	*interruptible sleep*, le processus est en attente (*wait*) et peut être interrompu.
`D`	*uninterruptible sleep*, le processus est en attente (*wait*) et ne peut pas être interrompu (probablement en I/O).
`I`	*idle kernel thread*, le processus est *idle* ce qui signifie qu'il est à la fois *uninterruptible sleep* et *no load*. Ce processus est en attente de travail et ne participe pas au calcul de la charge du système.[12]
`T`	*stopped by job control signal*, le processus est stoppé (par un signal `SIGSTOP`). Il redeviendra prêt lorsqu'il recevra un signal `SIGCONT`.

s (*state*)	**État du processus**
`t`	*stopped by debugger*, le processus est stoppé par un signal reçu d'un débogueur (*debugger*).
`X`	*dead*, le processus est terminé. Il a terminé son `run()`. Cet état n'apparait normalement pas car un processus terminé est détruit et n'apparait plus.
`Z`	*zombie*, le processus est terminé mais n'a pas été récupéré par son parent ni par *init*.

Le processus parent de tous les processus est **init** de *pid* 1.

Un processus peut être lancé en tâche de fond (*background*) en entrant la commande suivie d'une esperluette `&`.

Un processus peut-être stoppé en lui envoyant le signal `SIGSTOP` au moyen, par exemple, d'un `Ctrl-Z`. Il peut alors être relancé — continué — en le plaçant en tâche de fond (*background*) *via* `bg <job number>` par exemple.

Le coin des commandes

`ps`

`ps` liste les processus du système. Sans option[14], liste les processus associés à la console (`TTY`) courant.

- `u` liste les processus appartenant au compte utilisateur courant;
- `a` liste tous les processus associés à un terminal;
- `ax`, `-e`, `-A` liste tous les processus;
- `faux` liste tout des processus sans restreindre aux processus de *user* (`a`), sans restreindre aux processus sans TTY (`x`), les sélectionne par *uid* (`u`) et les affiche dans le format *full* (`f`). Ce format implique liste les processus sous forme d'arbre.

`kill`

`kill <pid>` envoie un signal au processus *pid*.

Même si les signaux les plus courants sont les signaux de mort ou de demandes de mort, il en existe d'autres. La liste s'obtient par l'option `-L` ou `-l`.

```
kill -L
```

Par défaut, *kill* envoie le signal `TERM` qui demande au processus de se terminer ; une demande de mort. Pour insister et envoyer le signal de mort `SIGKILL`, il faut le préciser:

[12] Voir ce kernel commit[13] (*consulté le 10 fév. 2021*)

[14] Cette commande est un peu particulière car elle accepte le « style UNIX » où les options peuvent être groupées et précédées d'un tiret (*dash*) « - », le « style BSD » où les options peuvent être groupées et ne doivent pas être précédées d'un tiret et le « style GNU » où les options sont au format long et sont précédées de deux tirets.

```
kill -SIGKILL <pid>
kill -KILL <pid>
kill -9 <pid>
```

Si vous avez stoppé un processus *via* `Ctrl-Z` (ou en lui envoyant le signal `STOP` par `kill -STOP <ipd>`), vous pouvez lui envoyez le signal `CONT` par un

```
kill -CONT <pid>
```

Le signal `SIGHUP` est aussi intéressant car il demande à certaines applications de relire leur fichier de configuration.

Attention, le *pid* peut valoir **-1**, dans ce cas, il signifie : tous les processus exceptés *init* et le processus *kill* lancé. En ce sens, la commande suivante est déconseillée :

```
kill -9 -1
```

`top` et `htop`

`top` et sa version *++* `htop` montre la liste des processus.
`q` pour quitter.

`[h]top` supporte la navigation avec les touches `Up`, `Down`, `Left`, `Right`, `PgUp`, `PgDn` et les commandes suivantes (extrait, cfr. *man htop*) :

- `Space` *tag* ou *untag* un processus;
- `U` *untag* tous les processus;
- `l` affiche les fichiers ouvert par ce processus (nécessite `lsof`);
- `t` *tree view* vue en arbre;
- `k` envoie un signal sélectionnable dans un menu;
- `u` affiche uniquement les processus d'un *user*;
- `M` trie par usage mémoire;
- `P` trie par usage processeur;
- `F` sélectionne une ligne et suit (*follow*) le processus si la liste est retriée par exemple;

Par défaut, `htop` montre

- `PID` le *process id* ;
- `USER` le *user*;
- `PRI` la priorité du processus (habituellement, la valeur de *nice* augmentée de 20);
- `NI` la valeur de *nice*,
- `VIRT` la taille de la mémoire virtuelle (*virtual*) du processus;
- `RES` la taille résident en mémoire du processus (text + data + stack);
- `SHR` la taille des pages partagées en mémoire (*shared*);
- `S` le statut de processus *state*;
- `CPU%` le pourcentage CPU utilisé;
- `MEM%` le pourcentage mémoire;
- `TIME+` le temp en click horloge utilisé par le processus;
- `Command` la commande complète du processus.

`pstree`

`pstree` montre des processus sous forme d'arbre.

```
pstree
pstree <user>
pstree -ph
pstree <user> -ph
pstree <pid> -ph
```

- `-p` montre les *pid* et désactive la vue compacte;
- `-h` surligne le procesus courant et ses parents

`pidof`

`pidof` donne le *pid* — ou les s'il y a

plusieurs instance du même programme — du processus dont le nom est donné.

```
pidof init
pidof bash
```

Tâches périodiques `cron`

Les systèmes linux disposent du programme `cron` capable d'exécuter un tâche à un moment donné. Chaque utilisateur dispose d'un fichier *crontab* et d'une commande associée permettant d"y accéder.

```
crontab -e
```

Cette commande permet d'éditer le fichier *cron* de l'utilisateur. Ce fichier a l'allure suivante:

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
```

Pour qu'une tâche particulière soit exécutée à intervalle régulier — par exemple tous les jours pendant la nuit pour faire un backup — le script correspondant à cette tâche doit se trouver dans ce fichier.

Au fichier *crontab* de chaque compte sont associés des fichiers *crontab hourly*, *daily* et *weekly*. Ces fichiers sont lancés et référencés dans `/etc/crontab`.

Certains programmes placent leur fichier *cron* dans `/etc/cron.d`.

Pour empêcher ou permettre des comptes à utiliser *cron*, *root* peut créer des fichiers `/etc/cron.allow` et `/etc/cron.deny` contenant les comptes utilisateurices pouvant utiliser ou non *cron*. Généralement, tous les comptes peuvent utiliser *cron* et ces fichiers ne sont pas créés.

Remarque : `cron` n'est pas le seul programme permettant d'effectuer des tâches au temps `t`, *systemd* et ses *timers* aussi bien que ce soit beaucoup moins répandu.

Gestionnaire de paquet `apt`

`apt` (*advanced package tool*) est l'outil de gestion des paquets *debian*.

Une distribution linux est plus qu'un simple noyau linux, c'est un ensemble cohérent de logiciels qui sont « distribués » par une « entité ». Cette entité peut-être une entreprise ou une communauté. Une distribution linux rassemble les logiciels en un ensemble **cohérent** mais également **stable** et offre un système de maintenance de ces logiciels.

Chaque distribution a ses particularités : l'usage (bureautique, serveur…), le matériel sur lequel l'installer, la facilité d'utilisation, les choix prédéfinis, le support… Il faudra donc à un moment **choisir** sa distribution linux. Pour le cours, le choix s'est porté sur ***debian***.

En fonction de la distribution choisie, le système de gestion des logiciels — les « paquets » — diffère. Les plus connus sont les systèmes basés sur les paquets `deb` et `rpm`.

Un paquet `deb` est une archive au format `ar` contenant elle-même deux archives ; `control.tar.gz` et `data.tar.gz` et un fichier de version (`debian-binary`). Les paquets debian se manipulent à l'aide de la commande `dpkg` pour une gestion bas niveau paquet par paquet. La gestion quotidienne de son système se fait quant-à elle grâce à `apt`[15].

Les paquets *debian* sont disponibles sur plusieurs sites et il est également possible d'installer et maintenir un **miroir** de l'un de ces sites. La première chose à faire est de configurer le miroir utilisé. Cela se fait dans le fichier `/etc/apt/sources.list` qui pourrait avoir l'allure suivante[16] :

```
deb http://ftp.be.debian.org/debian/ \
    buster main contrib non-free
deb-src http://ftp.be.debian.org/debian/ \
    buster main contrib non-free

deb http://security.debian.org/ \
    buster/updates main
deb-src http://security.debian.org/ \
    buster/updates main
```

- *buster* est le nom de la *release* stable à l'heure de la rédaction de ces notes[17].
- *main contrib non-free* seul *main* est nécessaire à l'installation du système, les autres valeurs peuvent être ajoutées pour accéder aux paquets *contrib* ou *non-free*. Les paquets *non-free* peuvent être utile pour un *driver* propriétaire particulier par exemple.

`apt`

- `apt update` met à jour la liste des paquets disponibles ainsi que leur version en local ;
- `apt list --upgradable` liste les pa-

[15] Historiquement, la gestion des paquets debian se faisait à l'aide des commandes `apt-get` et `apt-cache`. Ensuite la commande `aptitude` a été conseillée. Aujourd'hui, la commande `apt` suffit.

[16] Pour une configuration en belqique, choisir un miroir proche. Le miroir de son fournisseur d'accès est un bon choix. Si beaucoup de machines debian sont installées dans son entreprise maintenir un miroir local est sans doute une bonne idée.

[17] Il est conseillé de renseigner le nom de la *release* plutôt que `stable` pour éviter un saut de version inopiné lors d'une mise à jour du système. Chez debian, elles se nomment : Hamm, Slink, Potato, Woody, Sarge, Etch, Lenny, Squeeze, Wheezy, Jessie, Stretch, Buster et Bulseye.

quets qui peuvent être mis à jour ;

- `apt upgrade` met à jour le système en téléchargeant les paquets et en les installant sur le système ;
- `apt search <pattern>` fait une recherche dans la liste des paquets disponibles à la recherche d'un paquet correspondant au *pattern* ;

```
# apt search bind9
En train de trier... Fait
Recherche en texte intégral... Fait
bind9/stable,now 1:9.11.5[cut] \
    amd64  [installé]
  Serveur de noms de domaines internet

bind9-doc/stable 1:9.11.5[cut] all
  documentation de BIND
[cut]
```

- `apt install <paquet>` installe le paquet *paquet* et les paquets dont il dépend ;
  - `--reinstall` cette option demande de faire une réinstallation du paquet ;
- `apt remove` désinstalle un paquet. Ne désinstalle pas les dépendances ;
- `apt autoremove` désinstalle les paquets que ne sont plus nécessaires ;

Remarque : Lors de l'installation d'un paquet, toujours faire un *update* avant une installation.

En effet, la procédure d'installation va tenter d'aller chercher le paquet dans la version renseignée dans la liste des paquets disponibles **locale**. Si le paquet a été mis à jour sur le miroir, `apt` tentera de télécharger un fichier qui n'existe plus.

Démarrage du système, `systemd`

Au démarrage du système, les étapes suivantes sont exécutées :

- à la mise sous tension, le système charge le *moniteur*, anciennement `BIOS` et actuellement `uefi` qui fait la vérifications matérielles (CPU, mémoires, périphériques…) et initie le *bootstrapping*.

  Ce programme peut être interrompu par configurer le *boot* du système et certaines configurations matérielles ;

  uefi au contraire de *BIOS* peut résider sur une partition du disque, le firmware *uefi* peut donc lire et charger le code se trouvant sur une petite partition. C'est là que se trouve le code *uefi* de *grub*.

- le moniteur est configuré pour chercher du code sur certains périphériques dans un certains ordre (un disque dur, une clé USB…) ;

- chargement du code `grub2`[18].

  Il est possible d'interrompre *grub* et de passer des options au noyau sur lequel le système va booter.

- si c'est le choix de linux qui est fait dans *grub*, chargement du noyau sélectionné — éventuellement en deux étapes — et montage de la partition système `/` ;

- chargement de *systemd*, le « super dæmon » dont la responsabilité est de lancer tous les services prévus dans l'ordre qui va bien. C'est-a-dire en gérant les dépendances et l'ordre suggéré par *root*.

  Dans un système fonctionnant avec *systemd*, le processus de *pid* `0` appelé `init` est `systemd`.

  Dans un système fonctionnant avec *Sys V*, le processus de *pid* `0` appelé `init` est (était) le script exécutant le fichier `/etc/inittab` et lançant ensuite l'exécution des scripts se trouvant dans `/etc/rci.d` où `i` est le *runlevel* choisi au *boot* de la machine. *Sys V* a été abandonné au profit de *systemd* par debian avec *Jessie*.

systemd est un gestionnaire de système et de services pour linux. *systemd* est compatible avec les scripts d'initialisation *Sys V* et remplace *sysvinit*. Un des principaux avantages avancés pour *systemd* est qu'il permet la parallélisation.

Il s'organise par **unités** (*unit*) ; les services (*.services*), les points de montage (*.mount*), les périphériques (*.device*)… Ces unités sont définies dans un fichier. Ces fichiers se trouvent dans `/etc/systemd/`, `/lib/systemd/system/`… (cfr. `man systemd.unit`).

Les unités sont rassemblées pour former des **cibles** (*target*) ayant un sens ; par exemple la cible `graphical.target` ou `sound.target`. La configuration de ces cibles comprend ce dont elles ont besoin pour pouvoir être exécutées.

Au niveau de la sécurité des processus, *systemd* place chaque service dans un groupe de contrôle (*cgroup*) dédié au service. Ceci permet une bonne / meilleure isolation du système.

[18] Les *bootloader* linux sont, dans l'ordre d'apparition et de disparition : *LILO*, *Grub* et *Grub2*.

Le coin de la commande

`systemctl`

`systemctl` est la commande principale pour contrôler et gérer l'état du système, lancer, stopper des services.

```
# systemctl [options] command [unit]
```

- `status [pattern | pid]` sans option, affiche l'état du système sinon affiche l'état de la cible, de l'unité… ;

Exemples sans les retours de commandes

```
$ systemctl status
$ systemctl status bind9.service
$ systemctl status 9823
$ systemctl /dev/sda
```

Exemple de sortie du status de bind9

```
$ systemctl status bind9
● bind9.service - BIND Domain Name Server
Loaded: loaded (/lib/systemd/system/\
bind9.service;\
enabled; vendor preset: enabled)
Active: active (running) since Tue \
    2021-02-23 08:02:16 CET; 3 days ago
    Docs: man:named(8)
Main PID: 6792 (named)
Tasks: 7 (limit: 4915)
Memory: 16.4M
CGroup: /system.slice/bind9.service
        └─6792 /usr/sbin/named -u bind
```

- `--failed` affiche les services qui ont échoués ;
- `list-units [pattern]` affiche les unités que *systemd* a actuellement en mémoire[19] ;

```
$ systemctl list-units \
    --state=running
```

- `list-unit-files` affiche les fichiers correspondants aux différents services ;
- `start [pattern]`
  `stop [pattern]`
  `reload [pattern]`
  `restart [pattern]`
  `reload-or-restart [pattern]` lance, stoppe, demande de relire le fichier de configuration (*reload*), stoppe et relance ou reloade ou restart le service correspondant au *pattern*.
- `enable <unit>`
  `disable <unit>`
  `is-enabled <unit>`
  rend l'unité active ou inactive ou dit si elle l'est, au *boot* du système ;
- `halt`
  `poweroff`
  arrête le système.

  La différence entre *halt* et *poweroff* est que la seconde mettra le système hors tension après l'arrêt du système.

  halt et *poweroff* sont des appels aux cibles `halt.target` ou `poweroff.tarqet` ;
- `reboot` *reboote* le système ;

La suite dans la page de manuel `man systemctl`.

[19] De même, il existe des commandes pour lister les *sockets* (`list-sockets`) et les *timers* (`list-timers`) dont nous ne parlerons pas dans ces notes.

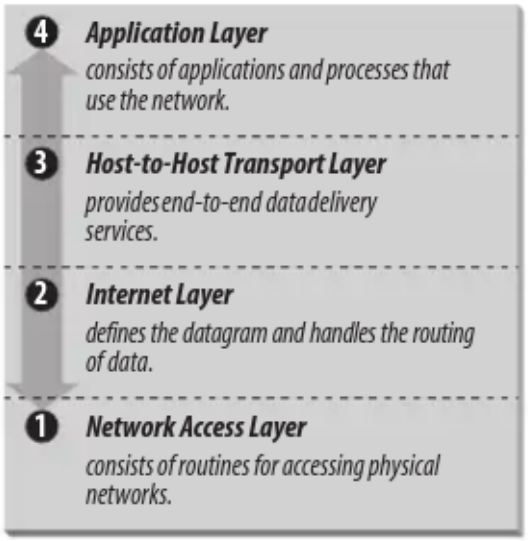
Rappel des bases réseau

Modèle **25**
Adresses **27**
Routage élémentaire **29**
Résolution d'adresse **29**
Noms **29**
Le coin des commandes **30**

Cette section se veut un rappel sur les concepts réseaux utiles au cours et à l'administrateurice[20] réseaux. Pour plus de détails, nous renvoyons le lecteur aux formations CISCO par exemple.

Modèle

En administration réseau nous nous contentons du modèle TCP-IP sous-modèle du modèle OSI.



Modèle TCP-IP (source : TCP/IP Craig Hunt)

- La couche application (*application layer*) concerne les applications utilisées sur le reseau : `http`, `smtp`, `ftp`, `telnet`…
- La couche transport (*host-to-host transport layer*) concerne le service de transport de données. Il y a deux protocoles :
  - **TCP** *transmission control protocol* service de transport de données avec détection et corrections d'erreurs, orienté connexion.

    TCP vérifie que le système distant est prêt à recevoir les données avant de les envoyer. Lorsque la poignée de main est faite, le système dit qu'il a établi la connexion.
  - **UDP** *user datagram protocol* service de transport de datagrammes sans connexion.

[20] Dans ces notes, l'écriture tente d'être inclusive. Un peu pour céder à la mode, un peu parce que ça « m'amuse » et un peu pour essayer d'inclure plus. J'essaierai de ne pas en abuser. Je troque généralement l'accord dit « du masculin l'emporte » contre l'accord de proximité et j'ajoute quelques nouveaux mots en évitant le point médian · qui fait peur ;-)

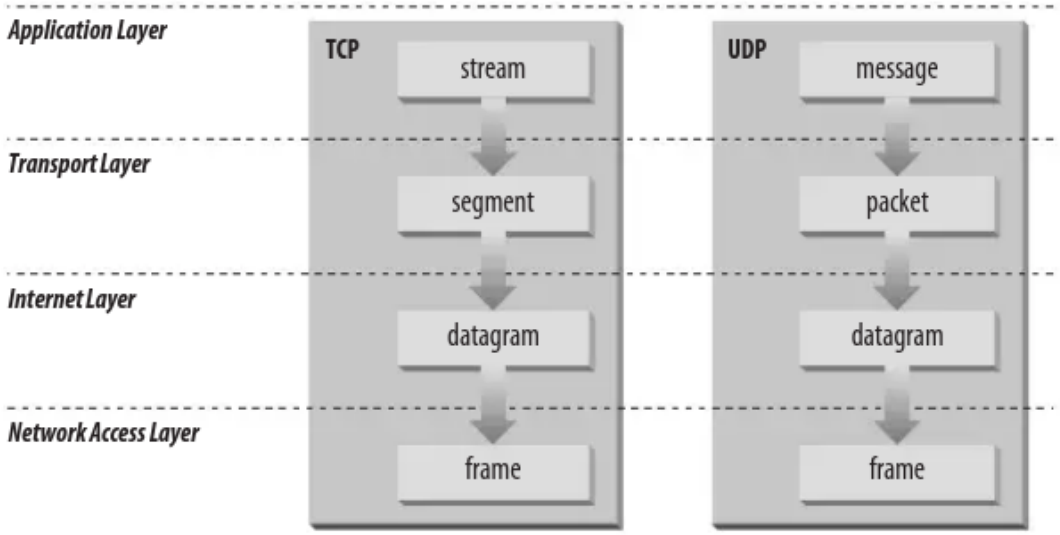


Figure : TCP et UDP dans le modèle TCP-Ip (source : TCP/IP Craig Hunt)

- La couche internet (*internet layer*) est la couche **IP**.

  IP est un protocole sans connexion. Ses rôles sont :

  – définir le datagramme (*datagram*) qui est l'unité de base pour la transmission sur internet;
  – définir le schema des adresses sur internet;
  – acheminer les données entre la couche *network* et la couche *transport*;
  – router les datagrammes;
  – s'occuper de la fragmentation et du réassemblage des datagrammes.

  ICMP *internet control message protocol* est un protocole inhérent à IP utilisant les datagrammes IP pour du contrôle, des rapports d'erreurs et de l'information.

  – *flow control* lorsque les datagrammes arrivent trop vite pour être traités, la destination ou une passerelle (*gateway*) intermédiaire envoie un *ICMP Source Quench Message* à l'expéditeur demandant d'interrompre l'envoi de manière temporaire;

  – *detecting unreachable destinations* lorsque le système détecte que la destination n'est pas atteignable, le système envoie un datagramme *Destination Unreachable Message*. Si la destination est un réseau ou un hôte, le message est envoyé par une passerelle intermédiaire. S'il s'agit d'un port inatteignable, c'est l'hôte qui envoie le message;

  – *redirecting routes* une passerelle envoie le message *ICMP Redirect Message* pour demander à un hôte d'utiliser une autre passerelle probablement parce que celle-ci est un meilleur choix. Ce message n'est envoyé que sur un même réseau.

  – *checking remote hosts* un hôte peut envoyer un message *ICMP Echo Message* pour tester si un sys-

tème distant est *up* et opérationnel. Lorsqu'un système reçoit ce message il (peut) y répondre en renvoyant le paquet. `ping` utilise ce message.

- La couche réseau (*network access layer*) est la couche **Ethernet**.

Adresses

Adresse IPv4

Une adresse IPv4 est une valeur de **32 bits** habituellement écrite comme 4 valeurs décimales séparées par un point « `.` » (*dotted decimal notation*) chaque nombre décimal représente 8 bits de l'adresse de 32 bits, chacun des nombres se trouvant dans l'intervalle 0-255.

Une adresse IPv4 se compose de deux parties :

- la première partie représente le numéro du **réseau** ;
- la seconde partie le numéro d'un **hôte** dans le réseau.

C'est le *masque* de réseau qui détermine **où** l'on coupe l'adresse en deux. Ce masque est composé d'un nombre de bits à 1 suivi d'un nombre de bits à 0.

Un réseau `/8` — anciennement, de classe A — a les 8 bits de gauche — les bits de poids forts — à 1. Son masque (*netmask*) est `11111111000000000000000000000000` en base 2 ou `255.0.0.0` en base 10 et en regroupant les bits par paquets de 8 (par byte).

Par exemple, l'IPv4 `10.0.0.1/8` représente :

- le réseau `10` ;
- la machine `0.0.1`.

Dans ce reseau numéro 10, il est possible d'avoir 16_777_214 hôtes (2^24-2). Les adresses `10.0.0.0` et `10.255.255.255` étant réservées respectivement pour le numéro de réseau et le *broadcast*.

L'adresse IPv4 de la boucle locale (*loopback*) est `127.0.0.1`.

Adresse IPv6

Une adresse IPv6 est une valeur de **128 bits** (16 bytes)[21]. La représentation hexadécimale regroupe les octets par 2 séparés par deux points « `:` ». Ce qui fait 8 groupes de 4 chiffres hexadécimaux. Par exemple :

```
2001:0db8:0000:85a3:0000:0000:ac1f:8001
```

Il est permis de remplacer `0000` par `0` et de supprimer des groupes nuls tant que l'adresse ne devient pas ambiguë. L'adresse précédente peut s'écrire :

```
2001:0db8:0:85a3::ac1f:8001
```

Les datagrammes IPv6 ont été simplifiés et ne comportent plus que 7 champs au lieu de 14 pour l'IPv4 ce qui accélère les traitements au niveau des routers.

[21] Ce qui fait 2^{128} adresses possibles, soit $\pm 3\,10^{38}$. Pour l'image, si la surface de la terre était recouverte d'ordinateurs ayant chacun, une IPv6, il serait possible d'allouer $7\,10^{23}$ adresses IP / m^2 (*source commentcamarche.net*).

Une adresse peut être un identifiant pour une interface (*unicast*) ou pour un ensemble d'interfaces (*multicast*). En ce sens, une interface peut avoir plusieurs adresses IPv6.

La notion de masque subsiste. Le préfixe est l'élément commun à toutes les adresses d'une même plage. La plupart du temps, le masque est `/64`.

Le *broadcast* est remplacé par le *multicast*.

Les types d'adresse IPv6[22] [23] :

- les adresses *unicast* désignant une destination unique (un hôte) :
  - la boucle locale (*localhost*) `::1/128` ;
  - l'adresse locale (*link local*) `FE80::/10` non routable utilisable au sein d'un réseau local ;
  - l'adresse publique (*global unicast*) `2000::/3`[24] ;
  - l'adresse privée (*unique local*) `FC00::/7` est l'équivalent des plages d'adresses privées en IPv4. Le 8e bit doit être positionné à 1 ce qui donne le préfique `FD00::/8` pour un réseau local ;
- les adresses *anycast* qui sont des adresses pour lesquels le chemin emprunté est au plus proche ou au plus efficient ;
- les adresses *multicast* `FF00::/8` remplacent les adresses de *broadcast* désignant potentiellement plusieurs destinations :
  - les 4 bits les moins significatifs du 2e byte (`FF0s::`) identifient la portée de l'adresse :

    `1` locale à l'hôte
    `2` locale au lien
    `5` locale au site
    `8` locale à l'organisation
    `e` globale

Il existe plusieurs techniques pour assigner une adresse en fixant l'identifiant d'interface (les 64 bits de poids faibles) :

- l'identifiant d'interface peut être fixé de manière arbitraire ;
- l'identifiant d'interface peut être configuré automatiquement :
  - déduit de l'adresse MAC (cfr. RFC 4862) en utilisant par exemple MAC EUI-64 (cfr. RFC 4291). Comme ces techniques exposent l'adresse MAC, elles sont déconseillées par IETF depuis 2017 ;
  - autoconfiguration basée sur une clé secrète et le préfixe reseau (cfr. RFC 7217)
  - autoconfiguration par un tirage pseudo-aléatoire (cfr. RFC 4941).

[22] *Internet Protocal Version 6 Address Space* https://www.iana.org/assignments/ipv6-address-space/ipv6-address-space.xhtml

[23] *Introduction aux adresses IPv6* https://cisco.goffinet.org/ccna/ipv6/introduction-adresses-ipv6/

[24] C'est bien `/3` et donc `200`. Les adresses unicast globales `2001::/16` peuvent être réservées et sont allouées par bloc `/23` à `/12` par l'IANA, *iana (internet assigned numbers authority), the global coordination of the DNS Root, IP addressing…* https://www.iana.org/ . L'IANA alloue aux RIR (*regional internet registry*) — RIPE NCC pour l'europe — qui allouent à leur tour au LIR (*local internet registry*) qui sont généralement également fournisseur d'accès. Les adresses `2002::/16` sont utilisées par *6to4* pour acheminer du trafic IPv6 vers IPv6 à travers IPv4. Les autres adresses sont réservées à un usage ultérieure. Ceci explique pourquoi les adresses unicast globales IPv6 sont de la forme `2001…`.

  - obtenu **dynamiquement** à partir de `DHCPv6`

ARP est remplacé par **ND** (*neighbor discovery*) et des messages `ICMPv6`.

L'adresse IPv6 de la boucle locale (*loopback*) est `::1`.

Routage élémentaire

Les passerelles routent les données à travers le réseau. Pour ce faire, les composants réseaux, les passerelles et les hôtes, doivent prendre des décisions de routage.

Pour la plupart des hôtes les décisions sont simples:

- si la destination est sur le réseau local, les données sont délivrées à l'hôte;
- si la destination est sur un reseau distant, les données sont transmises à la passerelle locale.

Une telle table de routage peut avoir cette allure en IPv4:

```
alice@harmony:~$ ip route
default via 192.168.1.1 \
    dev eth0 onlink
192.168.1.0/24 dev eth0 proto kernel \
    scope link src 192.168.1.11 \
    metric 100
```

Remarque: depuis le noyau 3.6, il n'y a plus de cache pour le routage IPv4.

Résolution d'adresse

L'adresse IP et la table de routage adresse un datagramme à un réseau cependant le datagramme doit passer par la couche physique. L'adresse IP doit être mappée à une adresse physique dépendante du réseau physique ; habituellement **ethernet**.

Le protocole faisant la traduction IP → Ethernet est **ARP** *address resolution protocol* pour IPv4 et **NDP** *neighbor discovery protocol* pour IPv6.

ARP maintient une table de correspondance entre adresse IP et adresse Ethernet ou adresse MAC. Lorsque ARP reçoit une demande de traduction d'une adresse IP, il regarde dans sa table. Si l'adresse s'y trouve, il la retourne sinon, ARP broadcaste un packet à tous les hôtes sur le segment. Le paquet contient l'IP concernée. Si un hôte a cette IP, il répond et la correspondance IP-Ethernet est cachée — au sens mise en cache — dans la table ARP.

Noms

Chaque système porte un nom, son nom d'hôte (*hostname*) qui associé au nom de domaine (*domainname*) devrait être unique.

Sur une machine linux, le nom d'hôte est renseigné dans `/etc/hostname`. Il peut être lu ou changé *via* la commande `hostname`.

Le nom de domaine est donné par la commande `dnsdomanname` pour obtenir le nom dns et `[yp]domainname` pour obtenir le nom de domaine au sens NIS/YP (*yellow pages*)[25].

La méthode recommandée pour positionner le **fqdn** (*fully qualified domain name*) est d'écrire un *alias* au nom d'hôte dans le fichier `/etc/hosts`. Par exemple :

```
127.0.0.1 harmony.example.org harmony
```

Par défaut le fichier `/etc/hosts` est lu avant de faire une requête DNS.

Le coin des commandes

`ip`

`ip` est la commande à tout faire pour la configuration du réseau[26]. Elle se présente comme `ip <object> [<command>]` par défaut, c'est la commande `show` (*alias* pour `list`) qui est exécutée. Les objets peuvent être abrégés.

`ip address` (ou `ip a`) pour l'adressage IPv4 ou IPv6.

```
ip a
ip -6 a
ip a show dev eth0
ip a delete 2001:0db8:85a3…/64 dev eth0
```

- montre toutes les interfaces réseaux (IPv4 et IPv6)
- montre toutes les interfaces reseaux en se limitant à IPv6
- montre l'interface *eth*
- efface l'adresse précisée de l'interface eth0

`ip route` (ou `ip r`) pour manipuler la table de routage.

```
ip r
ip r add default via 10.0.0.1 dev eth0
ip -6 r add 2001:db8:1::/64 \
    via 2001:db8:42::1 dev eth0
```

- montre la table de routage
- ajoute une route par défaut via `10.0.0.1` sur l'interface *eth0*
- ajoute une route pour le réseau `2001:db8:1::/64` via `2001:db8:42::1` sur l'interface *eth0*

`ip neighbour` (ou `ip neigh` ou `ip n`) pour manipuler les tables des voisins. Pour IPv4, il s'agit de la table ARP.

```
ip n
ip n flush dev eth0
```

- montre la table
- supprime les entrées de la table pour *eth0*

`hostname`

`hostname` montre ou positionne le nom d'hôte de la machine.

```
hostname
hostname --fqdn
```

- montre le nom d'hôte
- montre le nom d'hôte ainsi que le domaine

[25] NIS (*network information system*) et ses *yellow pages* ne sera pas abordé dans ces notes.
[26] Cette commande remplace `ifconfig` avec `ip address`, `route` avec `ip route` et `arp` avec `ip neighbour`.

`netstat`

`netstat` affiche les connexions réseaux, les tables de routage, les statistiques des interfaces…

Voici quelques options[27] :

- `-r`, `--route` permet l'affichage des tables de routages
- `-i`, `--interfaces` montre une table de toutes les interfaces réseau
- `-6` par défaut les commandes sont en IPv6, cette option demande l'IPv6
- `-p`, `--programs` affiche le nom et le *pid* des processus propriétaire de chaque socket
- `-a`, `--all` affiche tous les sockets (par défaut connectés)
- `-e`, `--extend` donne quelques infos supplémentaires
- `-c`, `--continuous` pour afficher de manière continue
- `-l`, `--listening` affiche les sockets en écoute

```
netstat -rn
netstat -r6
netstat -p
netstat -lapute
```

- affiche les tables de routage IPv4 au format numérique
- affiche les tables de routage IPv6
- affiche tous les sockets ouverts
- affiche un peu tout[28]

`dig`

`dig [@server] [name] [type) [class]` est une commande pour l'interrogation d'un serveur DNS.

- `-4`, `-6` demande de faire la requête en IPv4 ou en IPv6
- `<type>` représente le type de requête demandé : A, AAAA, NS…
- `<name>` est le nom de la resource : `example.org`…
- `@server` précise le serveur DNS utilisé. Si aucun serveur n'est renseigné, c'est le serveur défini dans `/etc/resolv.conf` que est utilisé
- `+trace` fait la demande à partir du serveur de nom racine et montre les réponses de chaque serveur de nom.

```
dig
dig example.org
dig example.org MX
dig @127.0.0.1 +trace example.org
```

- montre les serveurs racines
- montre l'adresse IP associée au nom `example.org`
- montre le champ `MX` associé au nom `example.org`
- demande au serveur local l'adresse IP associée à `example.org` avec une trace des différentes requêtes.

[27] Exceptionnellement, je conseille de préférer l'aide en ligne (`netstat --help`) à la page de manuel.

[28] `netstat` est sans doute la commande pour laquelle moult moyens mnémotechniques existent dans lequel chaque personne peut faire son choix: `-alpe` `--ip`, `-taupe`, `-lapute`.

Le shell

Commandes `shell` 33
Les expansions 34
Les redirections 36
Les tests 36
L'historique 37
Les commandes internes de `bash` 38
Script `shell` 39
Astuces et raccourcis `bash` 40
Le coin des commandes 41
head 41
tail 41
grep 41

Le *shell* est l'interprèteur de commandes. C'est le programme faisant l'interface entre l'utilisateurice et la machine. C'est une interface en ligne de commandes accessible via une console et permettant à l'utilisateurice de lancer des commandes en entrant un texte.

Le premier *shell* `sh` est dû à Thompson Shell (1971) et a été remplacé par le shell de Stephen Bourne (1977). Brian Fox réécrit ce shell et l'appelle *borne again shell* `bash` en 1988. C'est le shell le plus répandu bien qu'il en existe d'autres: `csh`, `tcsh`, `ksh`, `zsh`…

Dans la suite, nous nous intéressons à `bash`.

`bash` peut être invoqué de différentes manières :

- comme shell de login (*login shell*), si le premier caractère de son argument 0 est un tiret « - » ou s'il est invoqué avec l'option `--login`;
- comme shell interactif (*interactive shell*) si l'entrée et la sortie standards sont connectées à un shell ou s'il est invoqué avec l'option `-i`[29]

Lorsque `bash` est lancé, il lit des fichiers d'initialisation :

- pour un shell de login (interactif ou non), le fichier `/etc/profile` — s'il existe — est exécuté ;
- pour un shell de login (interactif ou non), `bash` cherche dans l'ordre les fichiers `~/.bash_profile`, `~/.bash_login` et `~/.profile` et exécute le premier qu'il trouve (l'option `--noprofile` empêche ce comportement);
- pour un shell interactif (mais pas de login), `bash` exécute les fichiers `/etc/bash.bashrc` et `~/.bashrc` s'ils existent (l'option `--norc` empêche ce comportement);

Lors de l'*exit*, le fichier `~/.bash_logout` est exécuté s'il existe.

[29]La variable `$-` peut être consultée pour voir les options passées au shell courant.

Commandes `shell`

Une **commande simple** est une séquence optionnelle d'assignation de variables suivie de mots et de redirections éventuelles. La commande se termine par un opérateur de contrôle par exemple : `||`, `&&`, `&`, `;`, `|`, `|&`, `<new line>`… Le premier mot est la commande a exécuter et les mots suivants sont passés comme arguments à la commande.

Un *pipe* est une séquence de commandes séparées par le caractère *pipe* « `|` ».

```
command1 | command1
```

la sortie standard de la première commande `command1` est connectée à l'entrée standard de la deuxième commande `command2`.

Si les caractères « `|&` » sont utilisés, la sortie standard **et** la sortie d'erreur standard entre dans le *pipe*.

Un commande peut se terminer par « `&` ». Dans ce cas la commande est exécutée en tâche de fond (*background*) et le shell rend la main.

Les commandes peuvent être séparées par un ET « `&&` » ou un OU « `||` ».

```
command1 && command2
```

- `command2` sera exécutée si et seulement si `command1` retourne *true*.

```
command1 || command2
```

- `command2` sera exécutée si `command1` retourne *false*.

Une **commande composée** est l'une des constructions suivantes (la liste n'est pas complète, cfr. `man bash`) :

- `(list)` une liste de commandes entre parenthèses est exécutée dans un sous-shell ;
- `{ list; }` une liste de commandes exécutées dans le shell courant. Peut être vue comme un bloc ;
- `[[ expression ]]` évalue l'expression conditionnelle entre double crochets « `[[ ]]` » ;

```
$ [[ -a /tmp/file ]] \
  && echo "File exist"
```

  – affiche "File exist"… si le fichier existe.

- `for <name> in word… ; do <list>; done`

```
$ for i in $(seq 10); do echo $i; done
$ for i in one two three; do touch $i; done
```

  – affiche les valeurs de 1 à 10
  – crée les trois fichiers `one`, `two` et `three`

- `for (( expr1 ; expr2 ; expr3 )); do <list>; done`

```
$ for (( i=0 ; i < 10 ; i++ )) ; do
>   echo $i;
> done
```

  – affiche les valeurs de 0 à 9

- `if <list>; then <list>; [else <list>;] fi`

```
$ if [ $DISPLAY = ":0" ] ; then
>   echo "X is launch" ;
> fi
```

  – affiche "X is launch" si la variable `DISPLAY` vaut ":0"

`bash` permet de déclarer des **variables**.

```
name=value
```

Notons qu'il ne faut pas d'espace de part et d'autre du signe « = ».

Les **paramètres positionnels** sont les arguments du shell lors de son invocation : `$1`, `$2`…

Les **paramètres spéciaux** sont repris dans le tableau suivant :

Param	Description
`$*`	Tous les paramètres positionnels. Entre guillemets, représente un seul mot : `$1_$2_$3`… (où _ représente un espace)
`$@`	Tous les paramètres positionnels. Entre guillemets, représente plusieurs mots : `$1 $2 $3`…
`$#`	Nombre de paramètres (décimal)
`$?`	Code de retour de la dernière commande.
`$-`	Liste des options avec lesquelles le shell a été invoqué.
`$$`	*pid* du shell.
`$!`	*pid* de la dernière commande exécutée en arrière plan.
`$0`	Nom du script (ou du shell).

Les expansions

Toute une série d'**expansions** se font dans **cet ordre**:

1. expansion des accolades (*brace expansion*),
2. développement du tilde « ~ » (*tilde expansion*),
3. remplacement des paramètres et variables (*parameter and variable expansion*),
4. substitution de commandes (*command substitution*),
5. évaluation arithmétique (*arithmetic expansion*),
6. découpage des mots (*word splitting*) et
7. développement des noms de fichiers (*pathname expansion*).

1 L'**expansion des accolades** permet la création de chaines quelconques sous la forme d'un préambule facultatif, suivi de chaines entre accolades et séparées par des virgules (sans espace blanc), le tout éventuellement suivi d'un postambule.

```
$ echo a{b,c,d}e
abe ace ade
```

Pratique par exemple pour créer plusieurs répertoires `mkdir /tmp/dir-{one,twe}`.

2 Le **développement du tilde** de manière simplifiée se remplace par la valeur de `HOME`. Tous les caractères précé-

dent de premier slash « `/` » sont considérés comme un *login*.

```
$ echo ~
/home/alice
$ echo ~bob
/home/bob
```

3 Le **remplacement des paramètres** s'associe au symbole dollar « `$` » et aux accolades « `{}` ». Même si une variable peut s'écrire `$var` il est conseillé de l'écrire `${var}`

`${parametre}` est remplacé par la valeur du paramètre.

Le remplacement des paramètres est plus large que cette simple « valeur de variable » et les principaux remplacement sont repris dans le tableau suivant (cfr. *man pages* pour la totalité) :

Remplacement de paramètres	**Description**
`${parametre:-mot}`	Donne la valeur du paramètre. Si le paramètre n'est pas défini, donne la valeur de *mot*
`${parametre:=mot}`	Donne la valeur du paramètre. Si le paramètre n'est pas défini, donne la valeur de *mot* et initialise le paramètre avec la valeur de *mot*
`${parametre:?mot}`	Donne la valeur du paramètre s'il existe sinon affiche *mot* comme message d'erreur.
`${parametre:+mot}`	Donne la valeur du *mot* si le paramètre existe. Sinon ne retourne rien.
`${#parametre}`	Donne la longueur de la valeur du paramètre.
`${parametre:offset}`	Donne la valeur du paramètre jusqu'à *length* en
`${parametre:offset:length}`	commençant au caractère d'indice *offset*.
`${!prefix*}` `${!prefix@}`	Donne toutes les variables commençant par *prefix*.
`${parametre#mot}`	Supprime le schema correspondant au début.
`${parametre##mot}`	
`${parametre%mot}`	Supprime le schema correspondant à la fin.
`${parametre%%mot}`	
`${parametre/pattern/string}`	Substitue *pattern* par *string*.

4 La **substitution de commandes** s'associe au symbole dollar « `$` » et aux parenthèses « `()` »[30].

`$(command)` est remplacé par le retour de la commande.

La commande est exécutée dans un sous-shell.

5 L'**évaluation arithmétique** s'associe au symbole dollar « `$` » et à la double paires de parenthèses « `(())` ».

[30] La substitution de commande se fait également en utilisant les guillemets inverses (*back quotes*) « ` ». Il est conseillé de ne plus utiliser cette notation même si elle est encore fréquente dans la littérature.

`$((expression))` est remplacé par la valeur de l'expression arithmétique[31].

6 Le **découpage des mots** se fait à la suite des expansions précédentes et tous les mots sont séparés en fonction de l'espace blanc (sauf si `IFS` a été modifié). Pour que des valeurs ne soient pas séparées, il suffit de les entourer de guillemets « `"` ».

7 Le **développement des noms de fichiers** consiste à rechercher dans les mots les *jokers* éventuels : `*`, `?` et `[]`.

- `*` correspond à n'importe quelle chaine;
- `?` correspond à n'importe quel caractère. Un seul;
- `[…]` correspond à une suite de caractères, un intervalle ou à une classe de caractères.

Les redirections

Par défaut l'entrée d'une commande, est l'entrée standard — le clavier — et les sorties de la commande sont la sortie standard — le terminal. Ce comportement peut être modifié en **redirigeant** ces canaux[32].

`<mot` lit le fichier (ouvert en lecture) *mot* comme entrée standard.

`>mot` redirige la sortie standard dans le fichier *mot* (ouvert en écriture).

`2>mot` redirige la sortie d'erreur standard dans le fichier *mot* (ouvert en écriture).

L'utilisation de « `>>` » redirige une sortie en **ajout** dans le fichier.

Il est possible de rediriger deux sorties dans un même fichier. Dans ce cas, la première redirection redirige dans le fichier (`>mot`) et la seconde en faisant référence à la redirection précédente avec l'esperluette (`2>&1`).

Pour rediriger la sortie standard et la sortie d'erreur standard dans le même fichier :

```
$ ls > out 2>&1
```

Les tests

Les **expressions conditionnelles** sont utilisées pour tester les fichiers, les chaines de caractères et l'arithmétique. Ces conditions sont testées avec la commande « `[[ ]]` » ou « `[]` » ou encore la commande `test`.

`bash` peut évaluer des **expressions arithmétiques** par exemple lors de l'expansion arithmétique. Les opérateurs habituels — y compris ceux de post|pré-inc|décrémentation ainsi que l'opérateur conditionnel — sont disponibles.

Par exemple :

[31] Le format `$[expression]` est déprécié et va disparaitre dans les prochaines versions de `bash`. À oublier donc…

[32] Ces notes présentent la redirection des descripteurs de fichiers (*file descriptor*) 0, 1 et 2, l'entrée standard, la sortie standard et la sortie d'erreur standard. Il est possible de rediriger d'autres descripteurs de fichier. Voir `man bash`.

```
$ for (( i=0 ; i < 10 ; i++ )) ; do
>   echo $i;
> done
```

`bash` peut faire des tests sur les fichiers. Par exemple :

- `-a file` teste l'existence du fichier;
- `-d file` teste l'existence et si c'est un répertoire;
- `-f file` teste l'existence et si c'est un fichier régulier;
- `-r file` teste l'existence et si le fichier est *readable*;
- `-w file` teste l'existence et si le fichier est *writable*;
- `-x file` teste l'existence et si le fichier est *excutable*;
- `file1 -nt file1` teste si *file1* est plus récent (*newer than*) *file2*;

`bash` peut faire des tests sur les chaines de caractères. En voici quelques uns :

- `-n string` vrai si la longueur de la chaine est non nulle;
- `-z string` vrai si la longueur de la chaine est nulle;
- `string1 = string2` vrai si les deux chaines sont égales (`==` fonctionne aussi);
- `string1 != string2` vrai si les deux chaines sont différentes;
- `string1 > string2` vrai si les deux chaines sont triées lexicographiquement;

Exemples :

```
$ if [ -d /tmp ] ; then
>   echo "dir exist";
> fi
```

```
$ if test -d /tmp ; then
>   echo "dir exist";
> fi
```

```
$ VAR=file.txt
$ if test -z ${VAR} ; then
>   echo "foo";
> else
>   echo "bar";
> fi
```

L'historique

`bash` possède un historique des dernières commandes entrées. Par défaut cet historique mémorise 500 commandes. `echo ${HISTSIZE}` conserve cette valeur.

Les flèches haut et bas permettent de naviguer dans cette historique.

Au delà de cette utilisation basique, voici quelques usages de l'historique.

- `history` affiche l'historique. Associée à un `grep`, il est facile de retrouver une commande;
- `!n` réexécute la ligne `n`;
- `!-n` réexécute la ligne se référent à la commande courante - `n`;
- `!!` réexécute la dernière commande;
- `!string` réexécute la commande la plus récente commençant par la chaine `string`;
- `^string1^string2` répète la commande précédente en remplaçant `string1` par `string2`;

Il est également possible d'utiliser les évènements précédents autrement que de simplement les exécuter. Là où `!!` execute la dernière commande, `!!:$` représente le dernier mot de la dernière commande par exemple.

Derrière ce `:` peuvent se trouver d'autres « désigneurs » de mots (*word designators*). En voici quelques uns :

- `n` le `n`e mot de la commande ;
- `^` le premier ;
- `$` le dernier ;
- `i-j` du `i`e au `j`e mot ;
- `*` tous les mots sauf la commande elle-même

Par exemple `!!:$`. Cette notation peut être raccourcie : `!!:$` par `!$` et `!!:^` par `!^`, etc.

Encore derrière ces *word designators* peuvent se trouver des modificateurs (*modifiers*) également précédés de deux points « `:` ». En voici quelques uns :

- `r` retire le suffixe et laisse le nom de base (*basename*) ;
- `p` affiche la commande mais ne l'exécute pas ;
- `s/old/new` remplace la première occurrence de `old` par `new` ;
- `gs/old/new` remplace toutes les occurrences de `old` par `new` ;

Voici quelques usages[33] :

```
$ ls /etc/apache2/apache2.conf
$ vim !!:$
```

```
$ xpdf /elsewhere/book.pdf
$ vim !!:s/pdf/md
```

```
$ cp pam.conf pam.bak
$ vi !^
```

```
$ cp ~/longname.conf \
  /really/a/very/long/path/other.conf
$ chmod go-rw !!:2
chmod go-rw \
  /really/a/very/long/path/other.conf
$ ls -l !cp:2
ls -l /really/a/very/long/path/other.conf
-rw------- 1 bob bob 0 fév 24 10:22 \
  /really/a/very/long/path/other.conf
```

Les commandes internes de `bash`

La plupart des commandes linux sont des binaires ou des scripts répartis dans différents répertoires du *filesystem*. Il existe toutefois une série de commandes qui sont des **commandes internes** à `bash`. Ces commandes sont directement interprètées pas le shell sans lancer un processus spécifique exécutant la commande.

En voici quelques unes, pour la liste complète, consulter le manuel `man bash` :

- `source filename` lit et exécute les commandes du fichier *filename* ;

```
$ source ~/.bashrc
```

- `alias [name=value]`
  - sans argument affiche la liste des *alias* définis;
  - lorsqu'un argument est fourni, défini l'*alias*

```
alias ll="ls -l"
```

- `bg jobid` place le *job jobid* (liste accessible par `jobs`) en tâche de fond (*background*) comme s'il avait été lancé avec `&` ;

[33] Certains exemples sont issus de thegeekstuff[34].

```
$ mycommand
[Ctrl-z]
$ bg 1
```

- `cd [<dir>]` change de répertoire (*change directory*) vers le répertoire *dir* s'il existe, sinon, vers `HOME` ;

  `cd -` se rend dans le répertoire dans lequel l'*user* se trouvait précédemment. Pratique pour faire un aller-retour.

- `echo [-neE] [arg]` affiche *arg*. L'option `-n` supprime la passage à la ligne de fin, `-e` active l'interprètation des séquences d'échappement et `-E` la désactive. Cette option est l'option par défaut ;

- `exit [n]` quitte le shell avec la valeur de retour *n* si elle est fournée, 0 sinon ;

- `ft jobid` place le *job* en avant plan (*foreground*) ;

- `help [-dms] [pattern]` donne de l'aide sur une commande interne. L'option `-d` donne une description courte, `-m` donne l'aide dans un format *man page* et `-s` donne un usage court ;

- `history` affiche l'historique complet,
  `history -c` efface tout l'historique,
  `history -d offset` efface l'entrée en position *offset*,
  `history -d start-end` efface les entrées de *start* à *end*,

- `kill [-s sigspec | -n signum | -sigspec] [pid | jobspec]` envoie le signal *sigspec* ou *signum* au processus *pid* ou *jobspec*,
  `kill -l | -L` liste tous les signaux

- `pwd` affiche le réportoire courant ;

Script `shell`

Un script shell n'est qu'une suite de commandes shell placées dans un fichier.

Un script peut porter l'extension `.sh` ou `.bash` mais elle n'est pas obligatoire.

Un script shell commence par le nom du shell qui doit être utilisé. Cette première ligne, à l'allure suivante, s'appelle le ***shebang*** :

```
#! /bin/bash
```

C'est une bonne pratique de faire immédiatement suivre ce *shebang* d'un commentaire précisant la fonction du script, l'auteur… et puis une fonction en précisant l'usage. Par exemple :

```
usage() {
  echo -e "$(basename ${0}) options…"
  echo -e "…"
}
```

Une fonction se définit comme précédemment et s'appelle simplement en donnant son nom (sans les parenthèse).

```
usage
```

Un script shell se trouvant par exemple dans le fichier `yascript.sh` peut s'exécuter par :

- `bash yascript.sh`[35] ou;
- `./yascript.sh` si le fichier a été rendu exécutable au préalable.

[35] Lancé avec l'option `-x`, `bash` exécutera le script en mode *debug*.

Quelques options sont pertinentes pour écrire un script plus sécurisé.

`set -e` entraine que le script quitte après une commande en erreur, sans continuer.

```
badcommand
echo "end"
```

- comme la commande n'existe pas, *end* ne sera pas affiché.

`set -o pipefail` entraine que le code de retour d'un pipe ne soit pas celui de la dernière commande mais celui de la dernière commande en erreur.

Associée à l'option `-e`, permet au script de quitter dès qu'une erreur survient… même dans un *pipe*. Par exemple :

```
$ bash -c  "badcommand1
> | echo foo;
> badcommand2;
> echo bar"
foo
bash: badcommand1 : commande introu…
bash: badcommand2 : commande introu…
bar
```

```
$ bash -ceo pipefail  "badcommand1
> | echo foo;
> badcommand2;
> echo bar"
foo
bash: badcommand1 : commande introu…
```

`set -u` entraine que le script quitte s'il rencontre une variable qui n'existe pas.

Le nom du script et ses paramètres sont accessibles par `$0` pour le nom du script puis, `$1`, `$2`… pour les paramètres.

L'accès à une variable devrait se faire entre accolades « `{}` » **et** entre guillemets « `""` ».

Un script devrait | pourrait avoir l'allure suivante :

```
#!/bin/bash
#
# Demo script.
#
set -eo pipefail

readonly VAR=${1}

set -u

usage() {
  echo -e "$(basename ${0}) options…"
  echo -e "…"
}

# param verification
if [ -z "${VAR}" ]; then
  echo "error message…"
  usage
  exit 1
fi

# now, do effective stuff
```

Astuces et raccourcis `bash`

- `[Esc] t` intervertit les deux derniers paramètres d'une commande.

  Pratique lors de l'écriture de `systemctl apache2 start` au lieu de `systemctl start apache2`.

- Le paquet `bash-completion` propose une autocomplétion intelligente pour toutes les commandes. À intalle.

- `[Ctrl-r]<pattern>` affiche la première commande correspondant au schéma

(*pattern*) disponible dans l'historique.

- `[Esc] .` écrit la dernière commande (sans le réexécuter immédiatement).

Le coin des commandes

head

`head [-n]` affiche les n premières lignes d'un fichier. Par défaut, n vaut 10.

tail

`tail [-n]` affiche les n dernières lignes d'un fichier. Par défaut, n vaut 10. D'autres options sont disponibles (cfr. `man tail`)

- l'option `-f` affiche les dernières lignes en continu, pratique pour des fichiers de logs ;

```
# tail -f /val/log/syslog
```

grep

`grep [options] patterns [file…]` recherche chaque *patterns* dans chaque fichier *file*. `grep` peut lire sur l'entrée standard `-`. Voici quelques options (pour d'autres options, `man grep`) :

- `-i` *ignore case* ;
- `-v` inverse la sélection, donne les lignes qui ne correspondent pas au *pattern* ;
- `-c` n'affiche pas les lignes mais les comptes (*count*) ;
- `-m num` arrête de chercher après *num* lignes correspondant au *pattern* ;

Monitoring et fichiers de logs

DNS domain name system

La résolution de noms 43
1984, 2000, 2014 44
Fonctionnement d'un serveur DNS 45
Fonctionnement d'une requête 46
Une question de confiance 48
Les différents champs et les fichiers de zone 50
bind9 52
La configuration du *stub resolver*, le fichier resolv.conf 53
DNS menteur 53
Le coin des commandes 53
dig 54
Localiser une adresse IP 54

> There is no place like 127.0.0.1… perhaps ::1

Les adresses IP n'étant pas conviviales, nous retenons les noms de machines… un serveur de noms permet de faire la correspondance entre un nom d'hôte et une adresse IP.

La résolution de noms

Au commencement, les machines étaient peu nombreuses et rarement connectées à internet. La correspondance entre les noms de machines et les adresses IP se faisait dans le fichier `/etc/hosts`.

Un fichier *hosts* a l'allure suivante[36] :

```
$ cat /etc/hosts
127.0.0.1 localhost
127.0.0.1 harmony.in.esigoto.info \
    harmony

# The following lines are desirable
# for IPv6 capable hosts
::1     localhost ip6-localhost \
    ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Avec l'agrandissement des réseaux locaux, et l'augmentation des tables d'hôtes à maintenir sur ces machines, un service s'est chargé de recopier certains fichiers de configuration entre machines. Il s'agit de NIS (*network information server*). Les fichiers sont maintenus sur une seule machine, le serveur, et les clients interrogent ce serveur plutôt que de lire la version locale du fichier.

[36] Souvent un fichier *hosts* ne résout plus que la boucle locale et, éventuellement, des noms à des fins de tests.

Nous sommes dans les années 80, `NIS`[37] centralise et facilite la maintenance du fichier `/etc/hosts` mais également d'autres : `/etc/passwd`, `/etc/group`…

Plus tard, à partir de 1994, `DNS` lui sera préféré. En plus de mémoriser les noms de machines locales, `DNS` permet de faire la recherche pour tous les noms internet. Il permet la gestion d'un grand nombre de noms pas son système de dissémination de l'information et de mise à jour.

L'ordre dans lequel le système fait la résolution de noms se trouve dans le fichier `/etc/nsswitch.conf` et plus particulièrement avec l'entrée `hosts` qui peut avoir l'allure suivante par exemple :

```
hosts: files mdns4_minimal \
    [NOTFOUND=return] dns
```

- dans ce cas, le nom d'hôte sera recherché dans le fichier *hosts* puis *via multicast dns* et enfin, par une requête auprès du serveur DNS.

1984, 2000, 2014

Ces années marquent au sujet des noms de domaines.

Pour rappel, un nom de domaine est de la forme `example.org`. Une machine de ce domaine portera par exemple le nom `harmony.example.org`.

Les noms de domaine ont une structure hiérarchique. Chaque partie, appelée label, est séparée par un point. La partie la plus à droite est le domaine de premier niveau (*tld top level domain*) et doit être choisie parmi les *tld* existants.
Par exemple `org`.

La deuxième partie doit comporter entre 1 et 63 caractères et pour certains *tld* peut être accentuée.
Par exemple `example`.

Ces deux parties forment ce que l'on appelle habituellement le nom de domaine.
Par exemple `example.org`.

À ces noms de domaine peuvent être ajoutés d'autres label pour former des sous-domaines.
Par exemple `foo.example.org`, `bar.example.org`.

Au commencement toujours, les noms de domaine de premier niveau génériques (*gtld*) sont : `com`, `net` et `org` rapidement suivi de `int`, `edu`, `gov` et `mil`. Il s'agit des années 1984 et 1985.

Les années 2000 voient fleurir quelques nouveaux noms de domaines : `biz`, `info`, `aero`, `coop`, `museum`, `name`, `pro`, `jobs`, `travel`, `cat`, `mobi`, `asia`, `tel`, `xxx`, `post` et `sexy`. Il est encore possible de les recenser.

À ces noms de domaines s'ajoutent les noms de domaines de premier niveau nationaux (*cctld*) comme `be` pour la belgique.

Puis en 2014, c'est l'explosion, toute personne désirant un nom de domaine de premier niveau peut le demander et, moyennant finance, il sera disponible. À l'heure

[37] `NIS` s'appelle d'abord *yellow pages* et, bien après avoir été renommé, certaines commandes commencent encore par `yp`.

où je rédige, je compte 1551[38] nouveaux domaines de premiers niveaux (*new gtld*) là où ils étaient 22 dans les années 80.

La liste complète des noms de domaines de premiers niveaux se trouve sur le site de l'IANA.[39]

Fonctionnement d'un serveur DNS

> Serveur DNS ? Serveur DNS faisant autorité ou résolveur ?

Lors de l'interrogation d'un serveur DNS, soit le serveur connait la réponse à la question et la donne, soit il la cherche.

Le DNS peut connaitre la réponse à la question parce que celle-ci se trouve dans son **cache** ou parce que le serveur **a autorité pour la zone** concernée. Une information a une certaine durée de vie (*TTL*, *time to live*) déterminée par le serveur ayant autorité. Un serveur ayant autorité pour une zone (un domaine de premier niveau, un domaine, un sous-domaine) détient la liste des correspondances IP/nom et nom/IP pour la zone. Il peut communiquer l'information au serveur qui la demande. Si le serveur ne connait pas la réponse, il la cherche.

Il existe différents types de serveurs DNS :

- les **résolveurs** (*resolver*)[40] ou serveur DNS « à cache seul »,

  ces serveurs ont une bonne mémoire puisqu'ils ne peuvent répondre qu'avec les informations détenues en cache.

  S'ils n'ont pas l'information en cache, ils interrogent un serveur racine (voir ci-dessous) s'ils sont récursifs (*recursive*) ou un autre résolveur appelé *forwarder* s'ils ne le sont pas.

- les serveurs « faisant autorité » (*authoritative server*) pour une ou plusieurs zones,

  ces serveurs détiennent un fichier par zone[41] contenant les correspondances entre les adresses IP et les noms.

  Parmi ces serveurs ayant autorité, certains seront des **serveurs maitres** — détenant effectivement les fichiers de zone — et d'autres seront des **serveurs**

[38] Les noms des domaines de premier niveau commençant par `a` : .aaa .aarp .abarth .abb .abbott .abbvie .abc .able .abogado .abudhabi .ac .academy .accenture .accountant .accountants .aco .active .actor .ad .adac .ads .adult .ae .aeg .aero .aetna .af .afamilycompany .afl .africa .ag .agakhan .agency .ai .aig .aigo .airbus .airforce .airtel .akdn .al .alfaromeo .alibaba .alipay .allfinanz .allstate .ally .alsace .alstom .am .amazon .americanexpress .americanfamily .amex .amfam .amica .amsterdam .an .analytics .android .anquan .anz .ao .aol .apartments .app .apple .aq .aquarelle .ar .arab .aramco .archi .army .arpa .art .arte .as .asda .asia .associates .at .athleta .attorney .au .auction .audi .audible .audio .auspost .author .auto .autos .avianca .aw .aws .ax .axa .az

[39] *Root zone database* https://www.iana.org/domains/root/db

[40] Le terme résolveur peut prêter à confusion. Il ne faudra pas confondre le résolveur dans le sens de la partie du serveur DNS faisant la résolution de nom pour remplir son cache — le résolveur — et la configuration du résolveur du système qui précise quel est le serveur DNS à utiliser — le *stub* résolveur. Cette dernière configuration étant généralement faites dans le fichier `/etc/resolv.conf`. Nous y reviendrons.

[41] Ils en auront généralement deux par zone ; un pour la zone et la résolution nom → IP et un second pour la *zone inverse* et la résolution IP → nom.

esclaves qui obtiennent leurs fichiers d'un serveur maitre.

Ces deux aspects sont très différents et doivent être bien compris. Là où le résolveur donne la réponse contenue dans son cache, le serveur ayant autorité donne la réponse qu'il connait. La réponse qui se trouve dans un fichier de configuration ou autre. Les données ayant autorités doivent toujours être prioritaires sur les données contenues dans le cache. Si les deux services sont séparés, lors de l'interrogation d'un résolver, le client sait que la réponse n'a pas autorité et qu'elle a une durée de vie limitée. Elle n'est peut-être plus valable et il faudra un peu de temps — dépendant de la durée de validité de la donnée — pour obtenir la « bonne » valeur. Lors de l'interrogation d'un serveur ayant autorité, le client sait que l'information reçue est toute fraiche.

Dans ces notes, nous traitons avec `bind` qui est capable d'assumer convenablement les deux rôles.

Fonctionnement d'une requête

Un serveur DNS résolveur récursif a un fonctionnement **top/down** avec **cache**.

Détaillons le fonctionnement d'une requête DNS faites par un résolveur DNS récursif. Lorsqu'il ne connait pas la réponse à la question posée, il la cherche. Pour ce faire, le serveur interroge l'un des **serveurs racines** (*root servers*) — au hasard dans la liste qu'il détient. Aujourd'hui, la liste des serveurs racines est la suivante (extrait) :

```
$ cut /etc/bind/db.root
;       This file holds the information on root name servers needed to
;       initialize cache of Internet domain name servers
;       (e.g. reference this file in the "cache  .  <file>"
;       configuration file of BIND domain name servers).
;
;       This file is made available by InterNIC
;       under anonymous FTP as
;           file                /domain/named.cache
;           on server           FTP.INTERNIC.NET
;       -OR-                    RS.INTERNIC.NET
;
;       last update:    February 17, 2016
;       related version of root zone:   2016021701
;
; formerly NS.INTERNIC.NET
;
.                        3600000      NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.      3600000      A     198.41.0.4
A.ROOT-SERVERS.NET.      3600000      AAAA  2001:503:ba3e::2:30
;
;; --cut--
```

```
;
; OPERATED BY ICANN
;
.                       3600000      NS    L.ROOT-SERVERS.NET.
L.ROOT-SERVERS.NET.     3600000      A     199.7.83.42
L.ROOT-SERVERS.NET.     3600000      AAAA  2001:500:3::42
;
; OPERATED BY WIDE
;
.                       3600000      NS    M.ROOT-SERVERS.NET.
M.ROOT-SERVERS.NET.     3600000      A     202.12.27.33
M.ROOT-SERVERS.NET.     3600000      AAAA  2001:dc3::35
; End of file
```

Le serveur racine (*root server*) interrogé répond en renseignant l'IP — plutôt *une* IP — du serveur ayant autorité pour la zone de premier niveau concernée. Le serveur DNS requérant interroge alors cette nouvelle IP. Si le serveur interrogé à autorité, il répond, sinon, il renseigne le serveur ayant autorité pour le sous-domaine… et ainsi de suite jusqu'à la réponse. Une fois la réponse obtenue, le serveur DNS requérant conserve la réponse en cache pendant sa durée de vie (*TTL*).

Par exemple, pour une requête `pica.esigoto.info` faite *via* `dig +trace pica.esigoto.info` (extraits) et en vous aidant du schéma :

- demande au serveur DNS qui répond avec la liste des serveurs racines ;

```
.  3600000 IN NS D.ROOT-SERVERS.NET.
.  3600000 IN NS L.ROOT-SERVERS.NET.
.  3600000 IN NS F.ROOT-SERVERS.NET.
.  3600000 IN NS K.ROOT-SERVERS.NET.
.  3600000 IN NS M.ROOT-SERVERS.NET.
.  3600000 IN NS I.ROOT-SERVERS.NET.
.  3600000 IN NS E.ROOT-SERVERS.NET.
.  3600000 IN NS H.ROOT-SERVERS.NET.
.  3600000 IN NS J.ROOT-SERVERS.NET.
.  3600000 IN NS C.ROOT-SERVERS.NET.
.  3600000 IN NS B.ROOT-SERVERS.NET.
.  3600000 IN NS A.ROOT-SERVERS.NET.
.  3600000 IN NS G.ROOT-SERVERS.NET.
```

- demande à `D.ROOT-SERVERS.NET` qui répond avec la liste des serveurs ayant autorité pour la zone `info` ;

```
info. 172800 IN NS a0.info\
    .afilias-nst.info.
info. 172800 IN NS a2.info\
    .afilias-nst.info.
info. 172800 IN NS b0.info\
    .afilias-nst.org.
info. 172800 IN NS b2.info\
    .afilias-nst.org.
info. 172800 IN NS c0.info\
    .afilias-nst.info.
info. 172800 IN NS d0.info\
    .afilias-nst.org.
```

- demande à `b2.info.afilias-nst.org` qui répond avec la liste des serveurs ayant autorité pour la zone `esigoto.info` ;

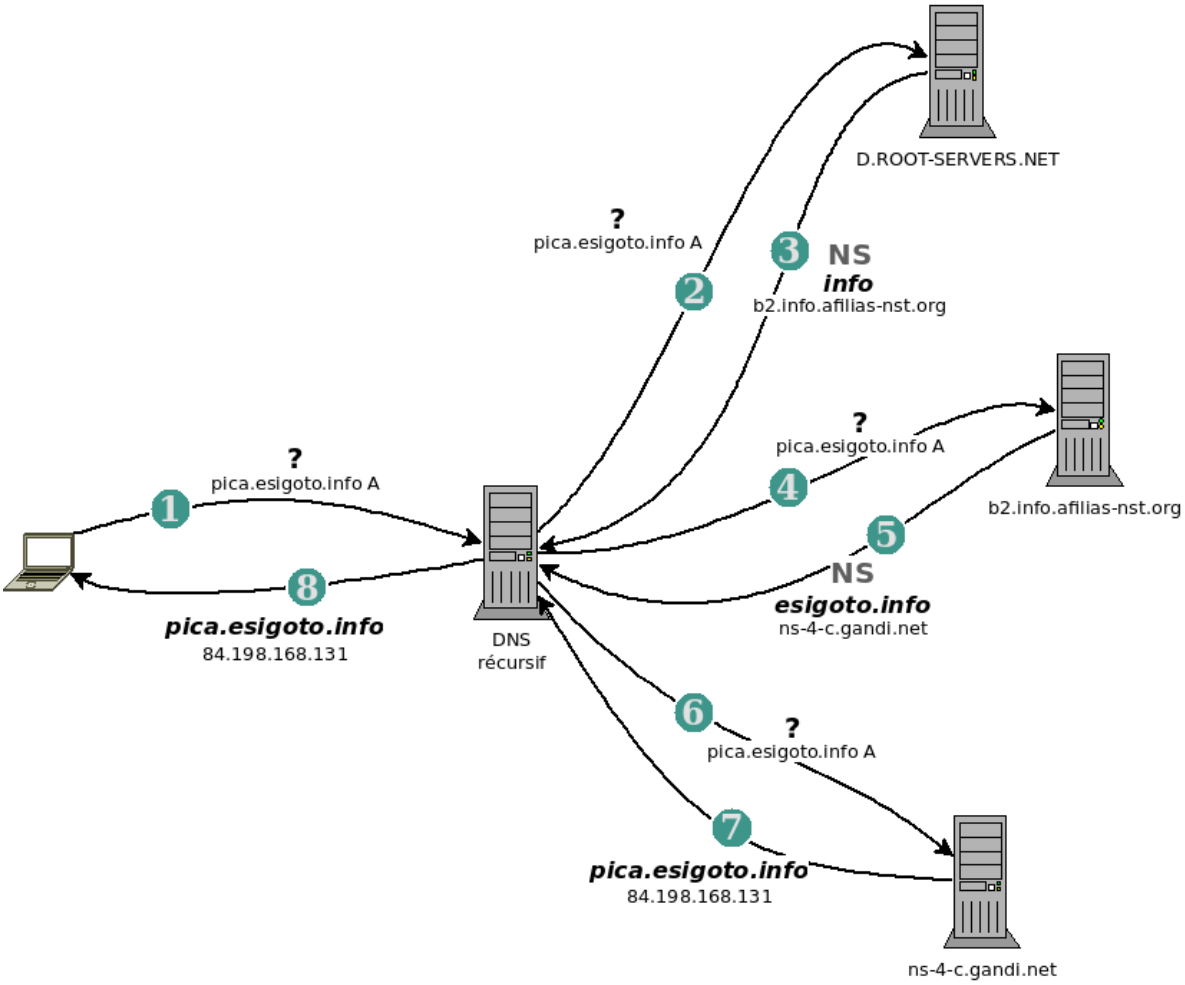
```
esigoto.info. 86400 IN NS \
    ns-4-c.gandi.net.
```

```
esigoto.info. 86400 IN NS \
    ns-167-b.gandi.net.
esigoto.info. 86400 IN NS \
    ns-83-a.gandi.net.
```

- demande à `ns-4-c.gandi.net` qui répond avec l'adresse IP de l'hôte *pica* car il a autorité pour la zone `esigoto.info` et connait donc la réponse ;

```
pica.esigoto.info.  1800  IN  \
    A  84.198.168.131
```

- la requête est bien *top/down* et le serveur requérant conservera en cache l'information pendant sa durée de validité.



Un résolveur *forwarder* quant-à lui fera la requête auprès de son *forwarder* qui répondra si la réponse est dans son cache ou qui exécutera la requête sinon… *via* un autre *forwarder* ou de manière récursive en fonction de sa configuration.

Une question de confiance

Il y a deux aspects au sujet de la notion de **confiance** dans le service DNS :

- la confiance dans le serveur lui-même ;
- la confiance dans le réseau.

Confiance dans le serveur DNS

Éliminons de suite la confiance dans le logiciel lui-même qui dépendra bien sûr du logiciel choisi : *bind* dans ces notes et c'est pareil pour *unbound*. Pour le logiciel libre, à l'habitude, le code peut-être audité et il est donc possible de vérifier que le logiciel fait bien ce qu'il prétend. Accordons notre confiance à *bind* et à *unbound*.

Bien que quasi personne ne choisisse quel serveur il utilise, ce devrait être fait car un serveur DNS peut **mentir** ou **bloquer** certaines requêtes[42][43]. Un serveur DNS peut mentir pour gagner de l'argent, pour respecter les lois — ou se plier aux contraintes c'est selon — d'un état…

Lorsque l'on ne choisit pas son résolveur DNS, c'est le serveur DHCP qui le propose et, dans un réseau local, c'est la *box* qui sera résolveur DNS. Ce résolveur est généralement *forwarder* et le *forwarder*, est le résolveur DNS du fournisseur d'accès à internet (*FAI*).

Dans ces conditions quelle confiance accorde-t-on à son FAI ?
Quelles sont les raisons qui pourraient le faire mentir ou le faire bloquer des requêtes ?

Il est alors possible de choisir un résolveur public. Les plus connus aujourd'hui sont ceux de Google `8.8.8.8` et de Cloudfare `1.1.1.1`. Ces serveurs n'utilisent (probablement[44]) pas `bind`. Ils s'annoncent plus rapides puisqu'ils font plus rarement de requêtes récursives auprès des serveurs racines, ont de bonnes connections réseaux…

Ces serveurs me bloquent-ils ? Me mentent-ils ?
Quelles sont les informations, ou plutôt quelle quantité d'informations récoltent-ils ?
Quid de ma vie privée ? Que reste-t-il de la décentralisation du service ?

Il est enfin possible enfin, d'installer son propre résolveur. Pour sa machine ou pour son réseau local. Si beaucoup de personnes font ce choix, cela peut avoir un impact sur les serveurs racines dont la charge risque d'augmenter.

Bref, tout n'est pas blanc ou noir…

Confiance dans le réseau

Ceci étant dit, un autre aspect de la confiance dans ce service est notre intimité et les requêtes DNS circulent par défaut en clair sur le réseau. Je peux me connecter en `https` au site `example.org` et personne ne saura ce que je consulte hormis le propriétaire du site mais *tout le monde* saura que j'y accède puisque ma demande de résolution de nom circulera en clair.

Si mon résolveur est sur ma machine, pas de problème, seul l'administrateurice aura accès aux requêtes qui sont effectuées… mais pas que. Le résolveur se trouvant sur ma machine va sortir ses requêtes en clair et une personne à l'écoute de mon réseau pourrait les voir.

Ne parlons pas de l'utilisation d'un serveur public. Le chemin est long entre

[42] *Quand ton serveur DNS te bloque ou te ment* https://blog.namok.be/?post/2016/10/18/quand-ton-serveur-dns-te-bloque-ou-te-ment

[43] *Mise en place d'un DNS menteur* https://blog.namok.be/?post/2017/03/05/mise-en-place-dns-menteur

[44] Probablement pas puisque ces services ne documentent pas vraiment.

ma machine et `1.1.1.1` par exemple.

Deux solutions existent pour remédier à cet aspect :

- **DNS sur TLS** (*dot*) encapsule les requêtes DNS dans une connexion chiffrée avec TLS ;
- **DNS sur HTTPS** (*doh*) encapsule les requêtes DNS dans une connexion chiffrée avec HTTPS.

L'avantage de la seconde sur la première est aussi un désavantage, elle permet l'utilisation de `HTTPS` qui est généralement disponible *un peu partout*. Les ports sont généralement laissés ouverts. Pour le reste, les concepts sont identiques.

Les serveurs publics tels que *Cloudflare* par exemple proposent ces services.

doh et *dot* peuvent être mise en œuvre à l'aide de `dnsdist`… ce qui sort du cadre de ces notes.

Les différents champs et les fichiers de zone

La requête DNS par défaut est celle demandant l'adresse IP correspondant au nom, c'est une requête pour l'enregistrement `A`. Il existe différents types d'enregistrements disponibles sur le site de l'IANA.[45] Voici les principaux :

- `A` adresse IPv4 (*host address*)
- `AAAA` adresse IPv6 (*host IPv6 address*)
- `NS` serveur de nom ayant autorité (*authoritative name server*)
- `CNAME` alias (*canonical name for an alias*)
- `SOA` début du fichiers de zone (*start of a zone of authority*)
- `PTR` nom de domaine (*domain name pointer*)
- `MX` serveur de mail (*mail exchange*)
- `TXT` zone de texte (*text strings*)
- `DNSKEY` TODO (*dnskey*)
- `AXFR` transfert de zone (*transfer of an entire zone*)

Ce sont ces différents types d'enregistrements que contiennent les fichiers de zone. Un fichier de zone contient :

- l'enregistrement `SOA` précisant la zone, il est requis et c'est le premier enregistrement ;
- un enregistrement `NS` précisant le serveur de nom ayant autorité pour la zone, il est requis ;
- des enregistrements `MX` précisant le serveur de mail. Idéalement ils sont au moins deux ;
- des enregistrements `A`, `AAAA` et `CNAME` pour les zones ;
- des enregistrements `PTR` pour les zones inverses ;
- des enregistrements `TXT`,

  ces enregistrements contiennent différentes informations. Par exemple, des commentaires pour la validation d'un certificat *https*, des valeurs `SPF`, `DMARK`… pour la sécurisation et la paramétrisation des serveurs de mails…

La zone locale a cette allure :

[45] *Domain Name System (DNS) Parameters* http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml

```
$TTL    604800
@ IN SOA localhost. root.localhost. (
         2         ; Serial
    604800         ; Refresh
     86400         ; Retry
    419200         ; Expire
    604800 )       ; Negative Cache TTL
;
  IN NS      localhost.
@ IN A       127.0.0.1
@ IN AAAA    ::1
```

Comme illustration d'une zone plus générale, la zone `esigoto.info` ayant quelques services en IPv4 et en IPv6 pourrait avoir cette allure :

```
@ 10800 SOA ns1.gandi.net. hostmaster.gandi.net. (
    2021030501 2h 30m 30d 1h
)
@ 10800 IN MX 10 spool.mail.gandi.net.
@ 10800 IN MX 50 fb.mail.gandi.net.
@ 10800 IN A 91.121.216.124
@ 10800 IN AAAA 2001:41d0:8:52c9:0:ff:fee3:bd57

atacama 1800 IN A 84.198.168.129
atacama 1800 IN TXT "v=spf1 ip4:84.198.168.129 mx -all"

pica 1800 IN A 84.198.168.131
pica 1800 IN TXT "v=spf1 ip4:84.198.168.131 mx -all"
vlab 1800 IN CNAME pica
vlabesi 1800 IN CNAME pica

date 10800 IN CNAME momos.hipocoon.be.
imap 10800 IN CNAME access.mail.gandi.net.
paste 10800 IN A 91.121.216.124
paste 10800 IN AAAA 2001:41d0:8:52c9:0:ff:fee3:bd57
smtp 10800 IN CNAME relay.mail.gandi.net.
www 10800 IN A 91.121.216.124
www 10800 IN AAAA 2001:41d0:8:52c9:0:ff:fee3:bd57
```

- les commentaires du champ `SOA` ne sont pas présents. Les commentaires ne sont pas obligatoires ;
- les valeurs ne sont pas toutes données en secondes mais avec une unité. Par exemple `h` pour les heures ;

- deux champs `MX` sont proposés et une priorité est donnée aux serveurs de mails : l'une de 10 et l'autre de 50 ;

bind9

`bind` (*Berkeley Internet Name Daemon*) est sans nul doute l'implémentation la plus répandue du DNS. La première version date de 1984 et `bind9` est une réécriture des versions plus anciennes qui prend en charge *DNSSEC* par exemple.

`bind9` peut jouer le rôle de résolveur **et** de serveur DNS ayant autorité.

`unbound` quant-à lui est un résolveur DNS beaucoup plus récent (2004) pouvant avantageusement remplacer `bind` lorsqu'il n'est pas nécessaire d'avoir aussi un serveur DNS ayant autorité.

> Dans ces notes nous présentons essentiellement `bind9`.

L'installation de `bind9` est aussi simple que :

```
# apt install bind9
```

Les fichiers de configuration de `bind9` se trouvent dans `/etc/bind/`. Le répertoire a l'allure suivante (*debian buster*) :

```
$ tree /etc/bind
/etc/bind
├── bind.keys
├── db.0
├── db.127
├── db.255
├── db.local
├── db.root
├── named.conf
├── named.conf.default-zones
├── named.conf.local
├── named.conf.options
├── rndc.key
└── zones.rfc1918
```

Les fichiers `db.*` définissent les zones (cfr. ci-dessus). Le choix du nom de ces fichiers est libre.

`named.conf` est le point d'entrée de la configuration de *bind*. Chez *debian*, il contient des *includes* vers les fichiers `named.conf.*` :

- `named.conf.options` contient les options dans la section `options { }` dédiée.

  Principalement les interfaces sur lesquels le serveur va répondre. Par défaut le serveur DNS est local ;

- `named.conf.local` contient les zones pour lesquels le serveur a autorité (*master* et *slave*). Chacune dans une section *zone*, par exemple `zone`

[46] Dans les paquets récents de *bind*, ce n'est plus le fichier `db.root` qui est utilisé, mais le fichier `/usr/share/dns/root.hints`. C'est une bonne pratique de vérifier — à partir de `named.conf` quels fichiers de configuration sont utilisés. Ceci peut varier d'une distribution à l'autre.

`"example.org" { }` ;

- `named.conf.default-zones` contient les zones par défaut : le *broadcast* `db.0`, la zone inverse pour la boucle locale `db.127`, la zone inverse pour le *broadcast* `db.255`, la boucle locale `db.local`, la zone . `db.root`[46] ;
- `bind.keys` contient la clé *DNSSEC* pour les serveurs racines.

  FIXME développer cet aspect
- `rndc.key` contient le *hash* de la clé d'accès au serveur bind *via* rndc ;
- `zones.rfc1918` définit toutes les zones correspondant aux classes d'adresses privées comme étant vides.

Une fois la configuration faite, le service se gère à l'aide de *systemd via* la commande `systemctl` comme habituellement.

Au sujet des *logs*, ils se trouvent dans `/etc/syslog` dès lors que *bind* est configuré pour parler. Ceci peut se faire en ajoutant une section *logging* au fichier de configuration. Cette section peut avoir l'allure suivante (très verbeuse)_:

```
logging {
  category default { default_syslog; \
    default_debug; };
  category security { default_syslog;\
     default_debug; };
  category database { default_syslog;\
     default_debug; };
  category resolver { default_syslog;\
     default_debug; };
  category queries { default_syslog; \
    default_debug; };
  category unmatched { null ; };
};
```

La configuration du *stub resolver*, le fichier `resolv.conf`

Le résolveur — ou *stub resolver* — est un ensemble de routines de la librairie C qui donne accès au DNS. Les programmes susceptibles de faire appel à ces routines sont nombreux. Il s'agit de tous les programmes nécessitant une résolution de noms. Par exemple : `ssh`, `git`, `owncloud`, `dropbox`… mais surtout les navigateurs[47].

Le

`DNS` menteur

TODO

- rndc

Le coin des commandes

[47] Les navigateurs sont configurés par défaut pour pour utiliser le serveur DNS défini par le système. Pour un réseau local, ce serveur DNS est celui de la *box* (routeur-modem donnant accès à internet) et se réfère au serveur DNS du fournisseur d'accès à interenet (*FAI*). Les navigateurs modernes peuvent être configurés pour faire leurs requêtes DNS *via* HTTPS (*doh*). Dans ce cas, une configuration habituelle est d'utiliser un serveur DNS chez *Cloudflare*. Il en existe d'autres. Il est possible d'utiliser son serveur DNS local bien sûr. Dans un réseau d'entreprise, ce sont les administrateurices systèmes qui déterminent où sont faites les requêtes DNS.

dig

`dig` est la commande permettant d'interroger un serveur DNS.

Requête simple pour obtenir l'adresse IP correspondant au nom :

```
dig esigoto.info
```

Demande à un serveur racine la liste des serveurs racines (sans paramètre, c'est le comportement par défaut de `dig`) :

```
dig . NS @a.root-servers.net
dig
```

+trace

TODO transfert de zone

TODO - bind / unbound

Localiser une adresse IP

Il peut être intéressant de savoir à quelle région est attribuée une adresse IP, voire à quelle fournisseur d'accès (*FAI*). Pour ce faire, il faut utiliser un service tiers, par exemple ipapi.co[48]. Dans une console, une requête peut avoir cette allure :

```
$ curl https://ipapi.co/<IP>/yaml
$ curl https://ipapi.co/<IP>/yaml -s \
    | awk '/country\_name|region|org/'
```

1. version longue au format `yaml`;
2. version réduite au pays, à la région et à l'organisation éventuelle.

Todos

En vrac, les sujets à aborder

- les pages de manuels `man`
- boot (bios / ueefi, grub lilo… systemd)
- su / sudo
- ssh
  - échange de clé
- vérifier dns, db.root -> /usr/share/dns/root.hints
- ajouter commande pour retirer les commentaires dans un fichier de conf

```
grep -v "^[#|$]" /etc/apache2/apache2.conf | grep .
```

si dhcp, modifier /resolv.conf ne sert à rien puisqu'il est réécrit. mieux:

- installer le paquet *resolvconf*

  ```
  # apt install resolvconf
  ```

[48] //ipapi.co

- éditer le fichier `head` qui sera lu *avant* `/resolv.conf` et ajouter le nameserver local par exemple.

```
# cat /etc/resolvconf/resorv.conf.d/head
nameserver 127.0.0.1
```

parler de resolv.conf dans dNS

le résolveur ou la configuration du client

nameserveur

max 3 serveurs domain search

le coup du head

Index

adduser, 8
apt, 21
ARP, 27, 30
arp, 30

bash, 32
bind, 52

cd, 14
chage, 9
chgrp, 14
chmod, 14
cloudfare, 54
Ctrl-Z, 18

deluser, 9
df, 14
dig, 31, 54
DNS, 31
dns, 29
doh, 49
dot, 49
du, 14

editor, 3
Ethernet, 27

FAI, 49
filesystem, 10
fqdn, 30

grep, 41
group, 4
grub2, 23

head, 41
hostname, 29, 30
hosts, 43
htop, 18

ICMP, 27
ifconfig, 30
IP, 27
IPv4, 27
IPv6, 27

kill, 17

ls, 14

MAC, 28
man, 2
miroir, 21
mkfifo, 15
modèle TCP-IP, 25
mount, 14

netstat, 31
NIS, 43
nsswitch, 44

PAM, 9
passwd, 4, 7
pidof, 18
ps, 17
pstree, 18

resolv.conf, 31, 53
resolver, 45
rndc, 53
root, 3
routage, 29
route, 30

shadow, 4
shell, 32
SIGHUP, 18
signaux, 17
source.list, 21
state, 16
su, 9
systemctl, 24

tail, 41
TCP, 25
top, 18
touche, 15

umount, 14
UPD, 25

user, 4
useradd, 8
userdel, 9
usergroup, 9
yellow pages, 43
zone, 52

Bibliographie

Codutti, M(2003) : *Administration Système, ULB INFO151.*

Dawson, T, Purdy, G, & BAUTTS, T(s.d.) : *Administration réseaux sous Linux*3 ed.

Goffinet, F(s.d.) : *Introduction aux adresses IPv6.*

Hunt, G(s.d.) : *TCP/IP Administration des réseaux*3 ed.

(s.d.) : *Domain Name System (DNS) Parameters.*

(s.d.) : *Filesystem Hierarchy Standard.*

(s.d.) : *Filesystem Hierarchy Standard (pdf).*

(s.d.) : *iana (internet assigned numbers authority), the global coordination of the DNS Root, IP addressing….*

(s.d.) : *Internet Protocal Version 6 Address Space.*

(s.d.) : *Licence creative Common BY-NC-SA 4.0.*

(s.d.) : *Mise en place d'un DNS menteur.*

(s.d.) : *Quand ton serveur DNS te bloque ou te ment.*

(s.d.) : *Root zone database.*