
Démarrage et arrêt d'un système UNIX

Pascal Picard

Corto E.T.F., K&M

Sainte-Clotilde, Ile de la Réunion

Copyright © 2002-2005, 2006 Pascal PICARD, pascal@seth.homeunix.net

Permission to use, copy, modify, and distribute this documentation for any purpose with or without fee is here by granted, provided that the above copyright notice and this permission notice appear in all copies.

THE DOCUMENTATION IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS DOCUMENTATION INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS DOCUMENTATION.

1. Organisation du disque d'un PC
 2. Les étapes du processus de *Boot*
 - 2.1. Chargement et initialisation du noyau
 - 2.2. Détection et configuration des périphériques
 - 2.3. Processus systèmes
 - 2.4. Intervention de l'opérateur système (en mode manuel)
 - 2.5. Scripts de démarrage
 - 2.6. Opérations en mode multi-utilisateur
 3. Les étapes des scripts de démarrage
 4. Démarrage et arrêt d'un système GNU/Linux
 - 4.1. Démarrage avec *lilo*
 - 4.2. Niveaux ou *Runlevel*
 - 4.3. Le fichier `/etc/inittab`
 - 4.4. Scripts de démarrage *Debian*
 5. Démarrage et arrêt d'un système *BSD*
 6. Arrêt du système [shutdown] et redémarrage [reboot]
- Références

1. Organisation du disque d'un PC

Le disque dur d'un PC peut contenir différentes partitions, chacune pouvant accueillir un système d'exploitation. Cependant, à un instant donné seule l'une d'entre elles peut être active, elle correspond à celle chargée en mémoire au démarrage du PC.

Le premier secteur du disque contient une zone particulière appelée *MBR* [*Master Boot Record*] lequel décrit le schéma de partitionnement du disque. Cette zone est à l'origine du chargement du système quel qu'il soit.

Ce *MBR* d'une taille de 512 octets, est structuré ainsi :

- les 446 premiers octets contiennent le *loader* (chargeur) qui doit charger le programme de chargement du système d'exploitation à lancer.
- les 64 octets suivants décrivent les partitions en termes de taille, de localisation, de type et de statut. Cette structure est limitée à quatre entrées contraignant à quatre le nombre de partitions dites primaires ou principales. Cette limite peut être dépassée en faisant d'une des partitions primaires une partition dite étendue qui peut contenir des partitions logiques.
- Enfin les 2 derniers octets contiennent le *magic number*, i.e. une valeur numérique qui permet à certains systèmes de vérifier la signature du secteur de boot.

2. Les étapes du processus de *Boot*

Au démarrage d'un PC, le programme de chargement situé dans le BIOS du système (stocké en ROM) est exécuté, son rôle est de charger le programme de chargement du *MBR*.

Il existe plusieurs programmes de chargement, citons *lilo*, le *FreeBSD boot loader*, *grub* ... Ces programmes peuvent démarrer tous les systèmes d'exploitation (OS) fonctionnant sur un PC. dans le cas le plus simple le chargeur lance l'OS de la partition marquée comme active. A l'extrême (plusieurs OS différents), le chargeur laisse à l'opérateur le soin de choisir le système qu'il veut lancer, puis charge le secteur de *boot* de l'OS choisi qui à son tour lance l'OS.

Le processus typique de *bootstrapping* (démarrage de l'OS) est décomposable en six étapes :

1. Le chargement et l'initialisation du noyau.
2. La détection et la configuration des périphériques.
3. La création de processus systèmesponnés
4. L'intervention éventuelle de l'opérateur système (en mode manuel).
5. L'exécution des scripts de démarrage.

6. Les opérations en mode multi-utilisateurs.

2.1. Chargement et initialisation du noyau

Le noyau est un fichier stocké sur disque et l'une des premières tâches du processus de démarrage est de le charger en mémoire.

GNU/Linux procède en deux temps :

- Dans la première étape, la ROM charge en mémoire centrale (RAM) un petit programme localisé sur disque.
- Dans la seconde, ce petit programme se charge de trouver et charger le noyau.

Le noyau détermine la quantité de RAM disponible, notamment pour ses structures internes dont certaines ont une taille statiquement fixée. Une certaine portion de cette mémoire est alors exclusivement réservée au noyau et ce dernier affiche sur la console le total de la mémoire trouvée et le total disponible pour les processus utilisateurs.

2.2. Détection et configuration des périphériques

Du point de vue conceptuel le noyau est une couche qui s'interface avec le matériel. La détection et la configuration des périphériques (matériels) disponibles est donc une tâche importante. Configurer son noyau c'est définir la configuration matérielle dans laquelle il s'exécutera.

Le noyau tente donc de localiser cette configuration attendue et recherche des informations supplémentaires en interrogeant directement les périphériques. Il imprime sur la console (et conserve) des informations quant à ces investigations.

A noter qu'il est très souvent possible d'activer dynamiquement un nouveau périphérique non prévu dans la configuration originale.

2.3. Processus systèmes

A ce stade le noyau va créer un certain nombre de processus qualifiés de spontanés, dans la mesure où il ne résulte pas du mécanisme standard de **fork**.

Le nombre et la nature de ces processus varient d'un système à l'autre. Avec GNU/Linux, il n'y a pas de processus visible de `pid 0`, le premier processus de `pid 1` (i.e. INIT père de tous les processus utilisateurs subséquents) est accompagné d'un certain nombre de processus spontanés gérant le noyau (`kflushd`, ...), la mémoire (`kswapd`, ...), les E/S (`kiod`, ...), la journalisation (`kjournald` pour `ext3`). Sous *BSD 4.4* (le père des *flavors xBSD*, tous les processus sont les héritiers d'un processus unique créé au démarrage du système. Ce processus ancêtre génère trois processus spontanés ^[1] qui existe toujours tant que le système est démarré :

- Le processus noyau *swapper* de `pid 0` responsable du transfert entre la mémoire principale (RAM) et la mémoire secondaire (SWAP) des processus quand l'espace en mémoire principale est saturé.
- Le processus noyau *pagedaemon* de `pid 2` responsable de l'écriture de parties de l'espace d'adresse (pages) d'un processus vers la mémoire secondaire dans le contexte de gestion de la mémoire virtuelle.
- Le processus utilisateur *init* de `pid 1`, premier processus à s'exécuter dans l'espace utilisateur (i.e. hors noyau) et ancêtre de tous les autres processus utilisateurs.

Lorsque ces processus *spontanés* ont été créés, l'étape de **bootstrapping** est terminée, néanmoins à ce stade la plupart des démons (processus systèmes) ne sont pas encore lancés, impossible par exemple de se connecter. La suite des opérations est prise en charge par le processus **init**.

2.4. Intervention de l'opérateur système (en mode manuel)

Si le système **boote** en mode **single user**, suite à l'activation d'un paramètre au démarrage, le noyau en avertit le processus **init**. Selon les systèmes, **init** peut éventuellement activer un sous-processus chargé de demander le mot de passe du *super-utilisateur*, puis si l'identification est réussie, **init** crée un shell de connexion.

Selon les systèmes (notamment Debian GNU/Linux, FreeBSD) seule la partition racine est montée en lecture seule. On exécute d'abord manuellement un **fsck** sur toutes les partitions, puis on remonte en écriture la partition racine, avant de remonter toutes les partitions restantes. Ensuite on exécute éventuellement les tâches qui ont requis ce mode. Lorsque le travail est terminé, on met fin à ce mode pour soit passer directement en mode multi-utilisateur (séquence CTRL-D, sous FreeBSD) soit redémarrer/arrêter le système (commande **shutdown**).

2.5. Scripts de démarrage

Il s'agit de scripts shell choisis et exécutés par le processus **init**. Nous étudions cela dans une partie ultérieure.

2.6. Opérations en mode multi-utilisateur

Après l'exécution des scripts de démarrage le système est quasi-fonctionnel, il ne manque que la possibilité de se *logger*. Chaque terminal (y compris la console) qui permet de se *logger* est contrôlé par un processus *getty* lancé par le processus **init**, ce dernier étant aussi responsable du lancement du gestionnaire de connexion en mode graphique (`xdm` ou `wdm` ou `gdm`...). A l'issue de cette phase le système est complètement opérationnel et le processus **init** continue de jouer un rôle important (rappelons par exemple qu'il hérite des processus orphelins, qu'il gère le processus d'arrêt, reboot et passage en mode mono-utilisateur, ...).

3. Les étapes des scripts de démarrage

L'exécution des scripts de démarrage *System V* ou *BSD* permet d'exécuter les tâches suivantes :

- l'attribution d'un nom de machine.

- la définition du *time zone*.
- la vérification de l'intégrité du système de fichiers (**fsck**) en mode automatique seulement.
- le montage du système de fichiers
- l'activation de(s) zone(s) de *swap*.
- le nettoyage du système de fichiers (suppression des fichiers du répertoire temporaire), la vérification des quotas disque ...
- la configuration des interfaces réseaux.
- le lancement des démons (programmes serveurs) locaux
- le lancement des démons réseau et le montage éventuels des partitions *NFS*.
- le lancement des processus *getty* permettant aux utilisateurs de se connecter au système.

4. Démarrage et arrêt d'un système GNU/Linux

4.1. Démarrage avec lilo

Il existe aujourd'hui, dans le monde GNU/Linux, principalement deux *boot loaders* : *lilo* et *grub*. Nous étudions dans la suite le premier qui est souvent le *boot loader* par défaut. Nous proposons en exercice l'étude de *grub*.

Le programme *lilo* est composé de deux programmes. Le premier (amorce *lilo*) peut être installé soit dans le MBR soit sur le secteur de *boot* de la partition Linux. Le second est localisé dans le fichier `/boot/boot.b`

La commande `/sbin/lilo` permet de l'installer en exécutant les instructions contenues dans le fichier de configuration `/etc/lilo.conf`. Chaque fois que le processus de *boot* doit être changé (par exemple, suite à la recompilation d'un nouveau noyau), il faut modifier la configuration de *lilo* (i.e. le fichier `/etc/lilo.conf`) et ré-exécuter la commande `/sbin/lilo` pour la prise en compte.

Voici un exemple de fichier `/etc/lilo.conf` pour la distribution *Debian* :

??? Indique si le chargeur *lilo* est situé dans le MBR ou sur le secteur de boot de la partition Linux.

 Ici dans le MBR.

??? Identifie les différentes images de noyaux Linux que peut charger *lilo*.

??? Spécifie le/les image(s) (secteurs de boot) du/des autres OS à charger.

L'installation du fichier de configuration précédent (`/etc/lilo.conf`), par la commande `/sbin/lilo` conduit à la sortie suivante :

```
# /sbin/lilo
Added Linux*
Added LinuxORIG
Added openBSD
```

Enfin, notons qu'au démarrage, le chargeur *lilo* attend 2 secondes, avant de charger le noyau `/vmlinuz`. On peut modifier ce déroulement au moment de l'affichage du prompt **LILO**: en appuyant sur la touche <TAB > qui permet de lister les options de *boot* possible.

4.2. Niveaux ou *Runlevel*

Le système GNU/Linux^[2] a l'image de la famille *Unix System V* propose différents niveaux de fonctionnement (ou *run levels*). A un moment donné, un seul niveau peut être actif.

La commande **init** (associé au processus *init*) avec pour argument le niveau de fonctionnement, active l'ensemble des processus relatif à ce niveau.

Une même commande peut être associée à différents niveaux. Un niveau de son côté peut être une spécialisation du niveau précédent, i.e. il ajoute des services aux services déjà existant (par exemple le niveau 5 est une spécialisation du niveau 3). Un niveau peut aussi être complètement orthogonal au niveau qui le précède, ce qui se traduira par un mode de fonctionnement radicalement différent.

Unix System V défini traditionnellement huit niveaux de fonctionnement :

Tableau 1. Autres commandes

niveau	sémantique associée
0	hors-service. C'est le niveau utilisé pour arrêter la machine. La machine peut être débranchée sans problème.
1	maintenance système, en mode console pour l'opérateur root uniquement.
s ou S	mode mono-utilisateur

niveau	sémantique associée
2	mode multi-utilisateur. Système de fichiers montés, démons lancés (y compris réseau) sauf RFS [<i>Remote File System</i>], exemple NFS [<i>Network File System</i>].
3	mode multi-utilisateur. Système de fichiers montés, démons lancés (y compris réseau) plus RFS [<i>Remote File System</i>], exemple NFS ou CodaFS ... Ce niveau est une spécialisation du niveau 2.
4	niveau définissable par l'administrateur.
5	Etat micro-code, utilisé à des fins de maintenance et de diagnostics.
6	Etat <i>reboot</i> , i.e. mise hors service suivie du redémarrage.

La plupart des distributions GNU/Linux orientée *System V* ont adopté ces niveaux de fonctionnement. Certaines rajoutent même les niveaux 7,8,9 définissables par l'utilisateur. Le niveau 5 est traditionnellement associé au mode de connexion graphique avec le gestionnaire (par défaut) **xdm**. Enfin, les niveaux 1 et S (ou s) sont généralement indifférenciés, ce qui réduit à sept le nombre de niveaux. La distribution *Debian* n'utilise pas pour sa part les *run levels* 3,4 et 5.

4.3. Le fichier `/etc/inittab`

Le processus *init* contrôle généralement le démarrage en mode multi-utilisateur. Rappelons qu'au démarrage la tâche du chargeur [*loader*] est de charger le noyau en mémoire principale. Le noyau crée, entre autre, le processus *init* qui va analyser le fichier de configuration `/etc/inittab` pour déterminer les actions à accomplir.

Le fichier `/etc/inittab` est un fichier de type texte structuré en parties séparées par des commentaires. La syntaxe d'une ligne typique (hors commentaire) est la suivante :

```
id:runlevels:action:process:
```

ou :

- *id* : est un identificateur permettant d'identifier la ligne sans ambiguïté.
- *runlevels* : Liste le ou les niveaux dans le(s)quel(s) la commande doit être exécutée. Un argument vide signifie tous les niveaux sauf s (i.e. 0123456).
- *action* : permet de préciser le contexte d'exécution de la commande à exécuter. Les valeurs possibles sont les suivantes :
 - *initdefault* : spécifie le niveau de fonctionnement par défaut, généralement 2 ou 3. Si cette ligne n'existe pas, le système demande le niveau de démarrage souhaité sur la console.
 - *boot* : la commande est exécutée uniquement au démarrage du système, de plus *init* n'attend pas sa terminaison pour poursuivre son travail en analysant et exécutant les lignes suivantes du fichier.
 - *bootwait* : la commande est exécutée uniquement au démarrage du système, mais *init* attend sa terminaison avant de poursuivre son travail.
 - *once* : exécute la commande si elle n'est pas déjà lancée, sans attendre sa terminaison, pour passer à la suite.
 - *wait* : comme précédemment mais avec attente de la terminaison.
 - *respawn* : exécute la commande une première fois lorsqu'*init* analyse son fichier de configuration et réexécute la commande si elle se termine.
 - *sysinit* : exécuté une seule fois au démarrage à froid du système, sans tenir compte du niveau (i.e. tous niveaux). Réservées aux tâches fondamentales nécessaires au bon fonctionnement du système.
 - *off* : Si le processus associé à ce niveau s'exécute, le terminer. Permet aussi de désactiver les terminaux.
 - *power{wait,failnow,okwait}* : ...
- *process* : désigne la commande à exécuter.

4.4. Scripts de démarrage *Debian*

Au démarrage « normal » de *Debian*, le premier script exécuté par le processus *init* est `etc/init.d/rcS` (c.f. contenu du fichier `/etc/inittab` en section Section 4.3, « Le fichier `/etc/inittab` »), lequel exécute à son tour les scripts du répertoire `/etc/init.d`. Ils contiennent l'ensemble des commandes indispensables à l'initialisation et au contrôle du système. Globalement, ils permettent :

- **S05initrd-tools.sh** : démonte et libère le *RAM disk* initialisé par le *boot loader* lors du démarrage et utilisé par le noyau pour son chargement en deux temps.
- **S05keymap.sh** : chargement des préférences clavier (azerty, français par exemple).
- **S10checkroot.sh** : activation du swap, vérification et montage de la partition racine (*/*).
- **S18hwclockfirst.sh** : positionnement de l'horloge système en fonction de l'horloge matérielle en tenant compte du paramètre *UTC* (`/etc/default/rcS`).
- **S20modutils** : vérification des dépendances entre modules et chargement.
- **S30checkfs.sh** : vérification du reste du système de fichiers local.

- **S30etc-setserial** : configuration des périphériques séries si le fichier de configuration manuelle `/etc/serial.conf` existe, sinon la configuration automatique intervient plus tard, voir le script **S46setserial**.
- **S30procps.sh** :
- **S35mountall.sh** : montage du reste du système de fichiers local.
- **S35quota** : vérification des quotas disques.
- **S39dns-clean** : nettoyage du fichier `/etc/resolv.conf` (éventuellement modifié lors des connexions **ppp**).
- **S39ifupdown** : remise à zéro du fichier `/etc/network/ifstate` qui liste les interfaces réseaux actives.
- **S40hostname.sh** : attribution du nom machine, par lecture du fichier `/etc/hostname`.
- **S40iptables** : restauration des règles du *firewall*, si elles existent et ont été préalablement sauvegardées.
- **S41portmap** : démarrage du démon *portmap*, lequel active les services *RPC* [*Remote Procedure Call*] nécessaire par exemple au système de fichiers *NFS* [*Network File System*].
- **S45mountnfs.sh** : montage des partitions *NFS*.
- **S46setserial** : activation/restauration des périphériques séries (mode automatique).
- **S48console-screen.sh** : chargement des paramètres pour les consoles.
- **S50hwclock.sh** ajustement de l'horloge *CMOS* relativement au paramètre *UTC* (`/etc/default/rcS`).
- **S55bootmisc.sh** : tâches diverses à réaliser au démarrage (nettoyage des fichiers temporaires, effacement des verrous non libérés ...).
- **S55urandom** : initialisation du générateur de nombres aléatoires.
- **S70nviboot** : recouvrement des sessions éditeur *vi*.
- **S75sudo** : activation du programme **sudo**

Le processus *init* poursuit son exécution en lançant le script spécifique (`/etc/rc.d/rc 2`) du niveau de fonctionnement souhaité. Dans le cas d'une *Debian* c'est le niveau 2 (*initdefault*). Ce script lance à son tour les scripts contenus dans le répertoire `/etc/rc2.d`. Ces scripts sont en fait des liens symboliques sur d'autres scripts (scripts « maîtres ») placés dans le répertoire `/etc/init.d`.

Exemple 1. Un contenu du répertoire `/etc/rc2.d`.

```
S10sysklogd -> ../init.d/sysklogd
S11klogd -> ../init.d/klogd
S14ppp -> ../init.d/ppp
S18quotarpc -> ../init.d/quotarpc
S19amavis-postfix -> ../init.d/amavis-postfix
S20acct -> ../init.d/acct
S20bastille-firewall -> ../init.d/bastille-firewall
S20binfmt-support -> ../init.d/binfmt-support
S20inetd -> ../init.d/inetd
S20lpd -> ../init.d/lpd
S20makedev -> ../init.d/makedev
S20mysql -> ../init.d/mysql
S20nfs-kernel-server -> ../init.d/nfs-kernel-server
S20postfix -> ../init.d/postfix
S20postgresql -> ../init.d/postgresql
S20samba -> ../init.d/samba
S20ssh -> ../init.d/ssh
S20xfs -> ../init.d/xfs
S20xfstt -> ../init.d/xfstt
S21nfs-common -> ../init.d/nfs-common
S89atd -> ../init.d/atd
S89cron -> ../init.d/cron
S91apache -> ../init.d/apache
S91apache-ssl -> ../init.d/apache-ssl
S99fetchmail -> ../init.d/fetchmail
S99rmnologin -> ../init.d/rmnologin
S99wdm -> ../init.d/wdm
S99xdm -> ../init.d/xdm
```

Tous les scripts commencent soit par la lettre K (c.f. niveau de fonctionnement 6), soit par la lettre S suivi d'un numéro à deux chiffres puis d'un nom significatif. Cette convention permet d'ordonner les scripts dans l'ordre lexicographique ; ordre utilisé pour l'exécution.

Entrer dans un niveau de fonctionnement implique d'abord l'exécution des scripts commençant par la lettre K (appel des scripts « maîtres » avec l'argument *stop*) dont le rôle est d'arrêter les démons (services) puis l'exécution des scripts commençant par la lettre S (argument *start*), pour (re)démarrer les démons.

Les scripts « maîtres » sont en mesure de traiter l'argument qui leur est passé en paramètre, ce dernier prenant les valeurs classiques : *start*, *stop*, *reload*, *restart*, *force-reload*...

5. Démarrage et arrêt d'un système BSD

Un système *Unix* possède fondamentalement trois niveaux de fonctionnement :

- hors service, i.e. non allumé.
- mode mono-utilisateur.
- mode multi-utilisateur (mode de fonctionnement normal).

Nous avons vu au paragraphe Section 4.2, « Niveaux ou *Runlevel* » ajoute d'autres niveaux à ces trois niveaux fondamentaux, ce qui n'est pas le cas de la famille *BSD*.

La distribution *FreeBSD* exécute le script `/etc/rc` au démarrage. Son but est de vérifier et monter les système de fichiers, d'activer les interfaces réseaux, de configurer les périphériques ... Plutôt que d'utiliser un long script monolithique, les développeurs de cette distribution ont opté pour de petits scripts réalisant chacun une tâche bien définie. Ils sont appelés depuis le script « maître » `/etc/rc`. En voici la liste :

- `/etc/rc.atm` : configuration du réseau *ATM*.
- `/etc/rc.diskless1`, `/etc/rc.diskless2` : utilisé pour lancer un système *FreeBSD* sans disque.
- `/etc/rc.firewall` : activation du firewall (*ipfw*) pour la famille de protocole IPv4.
- `/etc/rc.firewall6` : activation du firewall (*ipfw*) pour la famille de protocole IPv6.
- `/etc/rc.i386` : script d'initialisation pour les systèmes basés sur l'architecture *i386*.
- `/etc/rc.isdn` : configuration et activation de(s) l'interface(s) *ISDN*.
- `/etc/rc.network` : activation du réseau en IPv4.
- `/etc/rc.network6` : activation du réseau en IPv6.
- `/etc/rc.pccard` : configuration de(s) l'interface(s) *PCMCIA*.
- `/etc/rc.resume` : gestion de l'évènement *APM resume*.
- `/etc/rc.sendmail` : démarrage du démon **sendmail**.
- `/etc/rc.serial` : paramétrage des périphériques sériels.
- `/etc/rc.shutdown` : pour l'arrêt des démons lors d'un arrêt ou lors de la transition mode multi-utilisateur vers le mode mon-utilisateur.
- `/etc/rc.suspend` : gestion de l'évènement *APM suspend*.
- `/etc/rc.syscons` : configuration des terminaux : type de clavier, *screensaver* ...
- `/etc/rc.sysctl` : positionnement des valeurs *sysctl*, par lecture du fichier `/etc/sysctl.conf`

La configuration du script « maître » intervient dans deux fichiers `/etc/default/rc.conf` et `/etc/rc.conf`. Le premier contient de très nombreux paramètres réglables [*tunable knobs* en terminologie *FreeBSD*] avec leurs valeurs par défauts. Le second contient un sous-ensemble de ces paramètres avec les valeurs déterminées par l'administrateur. La personnalisation des paramètres intervient toujours dans ce second fichier, puisque ses définitions ont pour but de redéfinir les valeurs par défaut. Une lecture attentive du *man section 5 de rc.conf* permet de tout savoir sur les paramètres à régler.

Exemple 2. Un exemple de fichier de configuration `/etc/rc.conf`

Tableau 2. Quelques paramètres de personnalisation

paramètre	sémantique associée
<code>hostname="Godel.corto.home"</code>	Nom de la machine, au sens du <i>DNS</i> .
<code>defaultrouter="192.168.200.1"</code>	adresse réseau de la passerelle.
<code>ifconfig_xl0="inet 192.168.200.2 netmask 255.255.255.0"</code>	Configuration de l'interface réseau, ici une carte ethernet 3Com™.
<code>ifconfig_xl0_alias0="inet 10.10.129.145 netmask 255.255.255.0"</code>	Permet d'associer une nouvelle adresse IP à la même interface.
<code>inetd_enable="NO"</code>	Désactive le « démon » inetd .
<code>sshd_enable="YES"</code>	Active le « démon » sshd .
<code>keymap="fr.iso.acc"</code>	Clavier de type <i>azerty</i> français.

6. Arrêt du système [shutdown] et redémarrage [reboot]

Les changements dans les tampons mémoires [buffers] ne sont pas immédiatement répercutés sur disque, on parle d'écritures asynchrones. Cela permet d'améliorer notablement les performances des opérations d'E/S. Mais le système est susceptible de perdre des données en cas d'arrêt imprévu (c'est moins le cas aujourd'hui pour les systèmes de fichiers journalisés).

Savoir arrêter ou *rebooter* son système correctement permet d'éviter bien des problèmes. Ils existent plusieurs méthodes :

- Utiliser la commande **/sbin/shutdown**
- Utiliser la commande **/sbin/halt** ou **/sbin/reboot**
- Utiliser la commande (GNU/Linux) **telinit** ou **init**.
- Utiliser la commande (GNU/Linux) **/sbin/poweroff**.

La commande **/sbin/shutdown** (GNU/Linux)

```
SYNOPSIS :  
  
shutdown [-t sec] [-arkhncfF] [time] [warning-message]
```

C'est la manière la plus sûre pour :

- arrêter proprement (-h) le système,
- rebooter proprement (-r) le système,
- passer en mode mono-utilisateur (option par défaut).

Cette commande prévient les processus en cours d'exécution en leur envoyant le signal de terminaison SIGTERM. Elle notifie au processus *INIT* le changement de *runlevel* :

- 0 [init 0] : arrêt.
- 1 [init 1] : mode mono-utilisateur, maintenance.
- 6 [init 6] : reboot.

Arrêter le système immédiatement [*runlevel 0*] : **/sbin/shutdown -h now**.

```
Arrêter dans 20 minutes [runlevel 0] :  
# /sbin/shutdown -h +20 "Arret du systeme, pour maintenance"  
  
Reboot à 12h30 (runlevel 6) :  
# /sbin/shutdown -r 12:30
```

La commande **/sbin/halt** (GNU/Linux)

```
SYNOPSIS :  
  
halt [-n] [-w] [-d] [-f] [-i]
```

Exécute les tâches essentielles pour l'arrêt du système (appelé par la commande **shutdown** avec l'argument -h) :

- termine les processus non essentiels.
- exécute l'appel système **call**.
- attend que toutes les écritures soient complètes.
- arrête le noyau.

```
Eléments d'administration système  
Arrêter le système immédiatement :  
# /sbin/halt
```

La commande **/sbin/reboot** (GNU/Linux)

```
SYNOPSIS :
```

```
reboot [-n] [-w] [-d] [-f] [-i]
```

Fonctionnement similaire à la commande **halt**, mais effectue un redémarrage [*reboot*].

```
« Rebooter » le système immédiatement :  
# /sbin/reboot
```

La commande **/sbin/init** (GNU/Linux)

La commande **telinit** est un lien symbolique sur **init**.

```
SYNOPSIS :  
  
init [-a] [-s] [-b] [-z xxx] [0123456Ss]
```

```
Passer en mode maintenance (mono-utilisateur)  
# /sbin/init 1
```

La commande **/sbin/poweroff** (GNU/Linux)

```
SYNOPSIS :  
  
poweroff [-n] [-w] [-d] [-f] [-i] [-h]
```

Fonctionnement identique à la commande **halt**, mais envoie une requête pour couper explicitement l'alimentation électrique sur les systèmes possédant un gestionnaire d'alimentation.

Références

[1] MAN. *Manual pages*.

[2] NEMETH Evi, SNYDER Garth, et HEIN Trent H.. *Linux Administration Handbook*. Prentice Hall PTR. 2002.

[1] Les flavors *BSD* héritière de la branche 4.4 génère d'autres processus [en fait des *kernel threads*] (*wmdaemon*, *bufdaemon*, *wnlru* ... sous FreeBSD, par exemple).

[2] A la notable exception de la distribution Linux Slackware, orienté *BSD*.