Administration Système sous Linux (Ubuntu Server)

Grégory Morel

2018-2019

CPE Lyon

Huitième partie

Tâches d'administration

Surveillance de l’activité du système

Encore quelques commandes utiles

`who` / `w`	qui est connecté / qui fait quoi
`last`	historique des connexions sur la machine
`uptime`	depuis quand la machine est allumée
`tload`	affiche la charge du système
`dmesg`	affiche les messages en provenance du noyau
`sysctl`	permet d'afficher et de configurer les paramètres du noyau (à chaud)
`hostname`	affiche le nom de la machine
`uname`	affiche des informations sur le système (version du noyau…)
`lshw`	liste le matériel
`lsusb`	liste les périphériques USB
`top` / `htop`	activité des processus en temps réel

Complément sur l’administration des utilisateurs

`/etc/issue` : contient le message affiché avant l'invite de saisie de login [1] :

```
Ubuntu 18.10 serveur tty1

serveur login: _
...

$ cat /etc/issue
Ubuntu 18.10 \n \l
```

`/etc/motd` : Message Of The Day : on peut éditer ce fichier par exemple pour prévenir d'un reboot de maintenance (évite d'envoyer *x* mails...)

1. Fichier `/etc/issue.net` : id. mais message affiché lors d'une connexion *à distance*

Aperçu de PAM

PAM *Pluggable Authentication Modules* : ensemble de modules permettant de mettre en place des mécanismes d'authentification avancés pour des outils nécessitant une sécurité renforcée.

Exemples de modules :

- authentification UNIX classique
- authentification LDAP
- authentification par empreinte digitale
- ...

Un outil fait appel à la bibliothèque `libpam.so` pour un besoin d'identification, et à un fichier de configuration dans `/etc/pam.d`

```
$ ldd /usr/bin/passwd
    ...
    libpam.so.0 => /lib/x86_64-linux-gnu/libpam.so.0
    ...
```

Format des fichiers de configuration PAM :

```
Type_module contrôle module arguments
```

Exemple :

```
$ cat login
#%PAM-1.0
auth    requisite    /lib/security/pam_securetty.so
auth    required     /lib/security/pam_env.so
auth    sufficient   /lib/security/pam_unix.so
auth    required     /lib/security/pam_deny
auth    required     /lib/security/pam_nologin.so
```

1. vérifie si root tente de se connecter depuis un terminal non sécurisé
2. charge l'environnement de l'utilisateur
3. tente une authentification via les mécanismes Unix classiques
4. retourne toujours faux
5. si le fichier /etc/nologin existe, il est affiché

Aperçu de PAM

Type_module :

`auth`	module d'authentification (login et mot de passe)
`account`	autorisation, gestion de comptes (est-ce que cet utilisateur est autorisé à exécuter ce service ?)
`passwword`	vérification et mise à jour des informations de sécurité (est-ce que le mot de passe est toujours valable ?)
`session`	modification de l'environnement de l'utilisateur

contrôle :

`required`	réussite requise ; en cas d'échec, les autres modules sont appelés, mais PAM retournera au final une erreur
`requisite`	en cas d'échec, l'authentification se termine immédiatement
`sufficient`	en cas de réussite, PAM retourne OK quoi qu'il arrive
`optional`	le résultat est ignoré

module :

`pam_unix.so`	authentification standard
`pam_env.so`	définition des variables d'environnement
`pam_securetty.so`	interdit une connexion root depuis un terminal non sécurisé (cf. `/etc/securetty`)
`pam_stack.so`	appelle un autre service PAM pour le chargement de modules supplémentaires
`pam_nologin.so`	interdit la connexion d'utilisateurs si le fichier `/etc/nologin` existe
`pam_deny.so`	retourne toujours un échec
`pam_console.so`	donne des permissions supplémentaires à un utilisateur local
`...`	...

Gestion des logs

Logs / journaux / fichiers de journalisation / historiques

Traces horodatées des actions du système et des services

Il y a des logs pour tout : le noyau (`kern.log`), le système (`syslog`), le gestionnaire de paquets (`apt/history.log`), le démarrage (`boot.log`)...

Fichiers texte, par convention placés dans `/var/log`

Analyser les logs = premier réflexe de l'administrateur système en cas d'anomalie

Logs / journaux / fichiers de journalisation / historiques

Traces horodatées des actions du système et des services

Il y a des logs pour tout : le noyau (`kern.log`), le système (`syslog`), le gestionnaire de paquets (`apt/history.log`), le démarrage (`boot.log`)...

- Fichiers texte, par convention placés dans `/var/log`

- Analyser les logs = premier réflexe de l'administrateur système en cas d'anomalie

logs

Logs / journaux / fichiers de journalisation / historiques

Traces horodatées des actions du système et des services

Il y a des logs pour tout : le noyau (`kern.log`), le système (`syslog`), le gestionnaire de paquets (`apt/history.log`), le démarrage (`boot.log`)...

- Fichiers texte, par convention placés dans `/var/log`

- Analyser les logs = premier réflexe de l'administrateur système en cas d'anomalie

logs

Logs / journaux / fichiers de journalisation / historiques

Traces horodatées des actions du système et des services

Il y a des logs pour tout : le noyau (`kern.log`), le système (`syslog`), le gestionnaire de paquets (`apt/history.log`), le démarrage (`boot.log`)…

- Fichiers texte, par convention placés dans `/var/log`

- Analyser les logs = premier réflexe de l'administrateur système en cas d'anomalie

logs

Le service spécialisé dans la gestion des logs est (sous Ubuntu) `rsyslog`[1]

Remplace les anciens `klogd` (logs du noyau) et `syslogd` (logs système).
Fonctionnalités notables :

- peut lire et étend les fichiers au format syslog
- gestion des dates plus complète (inclut l'année et peut aller jusqu'à la milliseconde)
- peut écrire les logs dans une base de données
- gère la rotation automatique des fichiers
- peut crypter (GSS-API, TLS) les messages lors de connexions à distance

1. Ca ne veut pas dire que tous les fichiers de `/var/log` sont gérés par rsyslog !

Fonctionnement

Rsyslog implémente le protocole `syslog` ; on utilise `logger` pour envoyer des messages au service syslog, et ainsi créer des entrées dans le fichier `syslog`

```
$ logger "log de test"
$ tail /var/log/syslog
...
Mar 25 11:43:25 serveur greg: log de test
```

On peut créer un “tag” pour identifier l’origine d’un message (par exemple le nom du script qui effectue le log), et on peut demander l’écriture du contenu d’un fichier dans le journal :

```
$ logger -t LOG_PERSO -f monFichier
$ tail /var/log/syslog
...
Mar 25 11:43:25 serveur LOG_PERSO: contenu du fichier
```

`dmesg` (= display messages)

Affiche les messages émis par le noyau au démarrage de la machine ou par la suite [1]

- tampon circulaire : au bout d'un certain nombre de messages, les premiers sont archivés par syslog

- dmesg lit le contenu du fichier `kern.log`, mais permet de filtrer ou mettre en forme certains éléments :

`dmesg -T` : affiche les dates dans un format lisible (vs. microsecondes)
`dmesg -lerr -k` : n'affiche que les messages d'erreur venant du noyau

1. Par exemple lors de la connexion de périphériques

The Journal

Nouveau système de logs, centralisé, introduit par `systemd` (gestionnaire de services); consultable via `journalctl`.

⚠ logs au format binaire[1] . Avantages :

- fonctions de recherche puissantes (filtrage par date, application, etc., un peu comme une base de données);
- intégrer des données binaires directement dans le journal (ex. : *dump* suite à un plantage)
- beaucoup plus difficile de modifier le journal (ex. : un intrus voulant effacer ses traces)

1. Un choix très controversé dans la communauté Linux!

Exemples :

- `journalctl -u cron` : seulement les messages de l'unité **cron**
- `journalctl _PID=1` : seulement les message du processus de PID 1
- `journalctl /usr/sbin/dhcpd` : seulement les messages provenant du daemon dhcp
- `journalctl -p err` : seulement les messages d'erreur
- `journalctl --since "today"` : seulement les messages du jour

Avec `journald`, **syslog** n'est plus nécessaire, mais peut tourner en parallèle

Netfilter

Netfilter

Module du noyau Linux ($\geq$ 2.4) permettant de filtrer et manipuler les paquets réseau qui passent dans le système

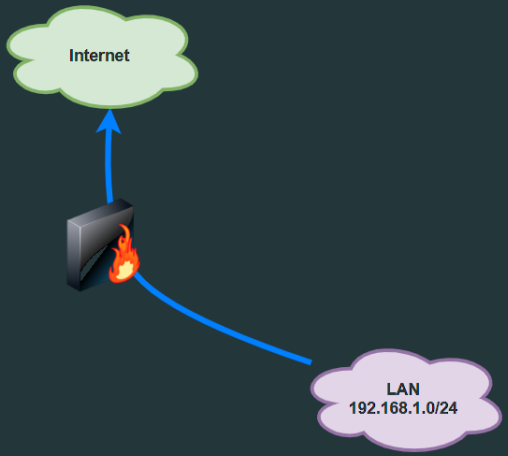
Pour info...

Il est prévu à terme que Netfilter soit (au moins en partie) remplacé par *nftables*, mais qui est encore en développement à ce jour.

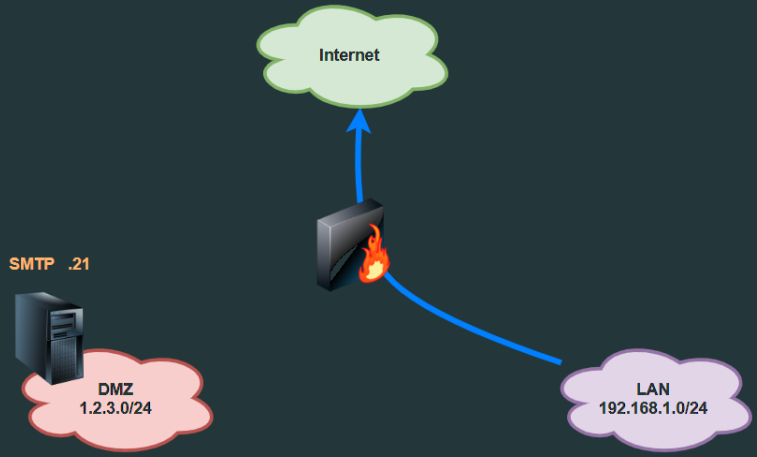
Netfilter est un *framework* ; il s'utilise via des utilitaires :

- iptables : bas niveau, pas toujours simple d'utilisation
- ufw (uncomplicated firewall) : alternative simplifiée à iptables
- Shorewall : une alternative à ufw

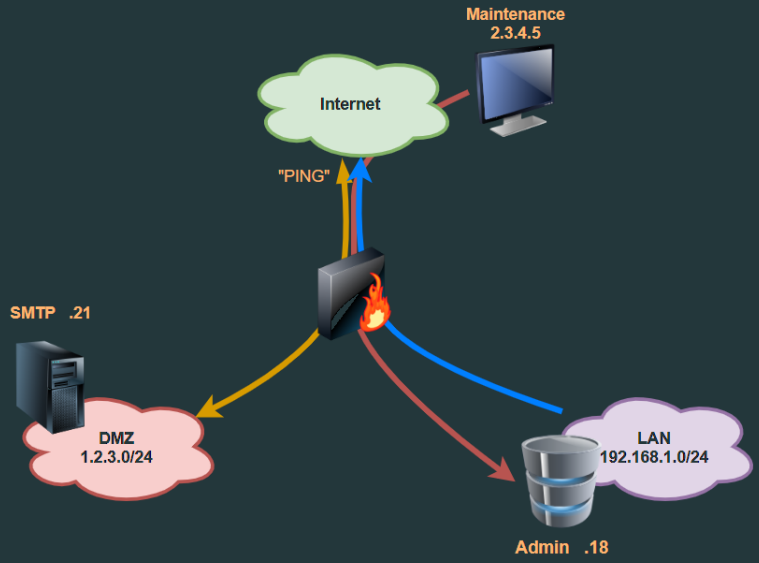
Netfilter



Netfilter



Netfilter


Maintenance
2.3.4.5
Internet
"PING"
SMTP .21
DMZ
1.2.3.0/24
LAN
192.168.1.0/24
Admin .18

4 règles à configurer pour ce réseau :

1. Autoriser les utilisateurs du LAN à accéder à Internet
2. Autoriser tout le monde à accéder au serveur mail
3. Autoriser le pare-feu à pinguer sur Internet
4. Autoriser une exception pour accéder au serveur sur le LAN

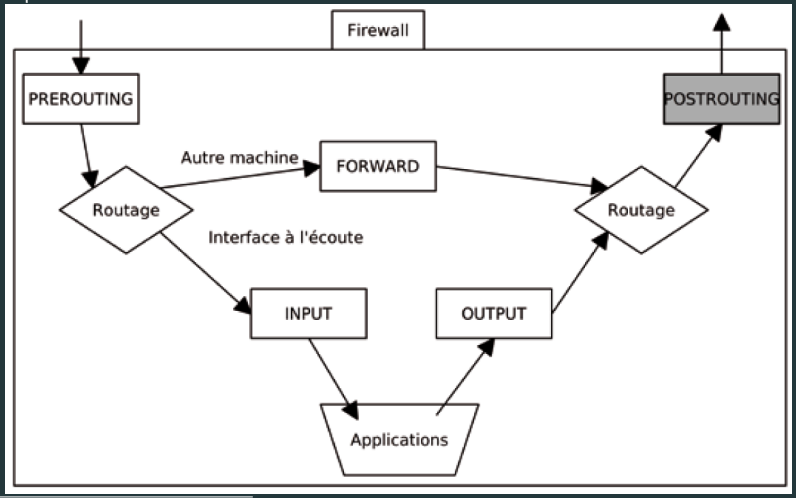
3 catégories :

paquets passant par le pare-feu (Règles 1 et 2) => `FORWARD`
paquets émis par le pare-feu (Règle 3) => `OUTPUT`
paquets à destination du pare-feu (Règle 4) => `INPUT`

On peut en rajouter 2 autres :

`PREROUTING` : traitement dès réception (ex. : modif. d'adresse de destination)
`POSTROUTING` : traitement avant émission (ex. : modif. d'adresse source)

Vie d'un paquet[1] :



1. Image : S. Rohaut

Principe de Netfilter : chaque paquet, entrant ou sortant, suit une ou plusieurs suites de règles appelées chaînes.

Principe des chaînes de règles

- Les règles sont lues dans l'ordre
- Dès qu'une règle est remplie, les suivantes sont ignorées
- Une règle est remplie si tous ses critères sont remplis
- Si un paquet passe toutes les règles, une règle par défaut peut être appliquée

Iptables

Pour gérer les chaînes, on utilise `iptables`.

- `iptables -vL` : liste les chaînes de règles
- `iptables -F` : supprime toutes les chaînes de règles ⚠
- `iptables -X` : supprime les chaînes définies par l'utilisateur

Lorsque le pare-feu n'est pas configuré, tout le trafic passe dans toutes les directions (`policy ACCEPT`) :

```
$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Exemple

Pour interdire tous les paquets en provenance de `192.168.1.11` :

```
iptables -A INPUT -s 192.168.1.11 -j DROP
```

- `-A INPUT` : ajoute une règle à la chaîne `INPUT`
- `-s` : source du paquet
- `-j DROP` : cible (*jump*) de la règle, si elle est satisfaite

Attention !

Lorsque la politique (*policy*) par défaut sur `INPUT` est `ACCEPT`, la dernière règle est souvent `-j REJECT` pour tout interdire sauf les règles précédentes. Avec `-A INPUT`, la nouvelle règle est placée à la suite et n'aura donc aucun effet.

Autres exemples

Pour interdire les entrées par `enp0s3` :

```
iptables -A INPUT -i enp0s3 -j DROP
```

Pour interdire le protocole ICMP (*ping*) en entrée :

```
iptables -A INPUT -p icmp[1] -j DROP
```

Pour interdire les connexions entrantes à destination du port 80 :

```
iptables -A INPUT -p tcp --dport 80 -j DROP
```

Pour interdire toutes les connexions sauf celle de 10.0.0.1 :

```
iptables -A INPUT -s ! 10.0.0.1 -j DROP
```

Pour loguer les événements :

```
iptables -A INPUT -m limit --limit 5/min -j LOG
--log-prefix "iptables denied: " --log-level 7
```

1. Voir `/etc/protocols` pour les autres protocoles

Iptables

Les chaînes sont regroupées en tableaux (ou *tables*). Il en existe 3 :

- `filter` : tableau par défaut; chaînes de filtrage pour accepter, refuser, ignorer un paquet
- `nat` : chaînes de modification des adresses IP ou des ports sources ou destinataires
- `mangle` : chaînes permettant de modifier certains paramètres à l'intérieur des paquets IP

Ex. : *MASQUERADING* (= NAT source) : autoriser les machines avec une IP privée à accéder à Internet

```
$ iptables -t nat -a POSTROUTING -s 192.168.1.0/24 -o enp0s3 -j MASQUERADE
```

Important

Les règles sont transmises dynamiquement au noyau, et sont perdues au redémarrage de la machine ! Il faut donc penser à les sauvegarder puis les restaurer.

iptables propose les commandes `iptables-save` et `iptables-restore`, mais qui sont peu pratiques.

Le moyen le plus simple est d'utiliser le paquet `iptables-persistent` :

```
$ sudo netfilter-persistent save
...
$ sudo netfilter-persistent reload
```

UFW : Uncomplicated Firewall

Front-end pour NetFilter

`ufw enable / disable`	Active / Désactive le pare-feu
`ufw status [verbose]`	Affiche le statut du pare-feu
`ufw allow / deny [règle]`	Autorise / Refuse une connexion
`ufw logging on / off`	Active / Désactive la journalisation
`ufw app list`	Liste les services qui ont des règles `ufw`
`ufw app info APP`	Affiche les règles de *APP*
`ufw [--dry-run] règle`	Affiche les changements impliqués par *règle* sans le

Exemples :

- `ufw default allow` : autorise le trafic entrant selon les règles par défaut
- `ufw deny 80` : bloque le port 80
- `ufw deny http` : bloque le service HTTP
- `ufw deny apache` : bloque le service Apache

Exécution de tâches en différé

at

`at` : exécuter une tâche en différé (une seule fois)

Exemple :

```
$ echo 'echo "Réunion Marketing"' | at 15:30 tomorrow
warning: commands will be executed using /bin/sh
job 1 at Wed Mar 27 15:30:00 2019
$ atq
1  Wed Mar 27 15:30:00 2019 a greg
$ atrm 1 ; atq
$
```

- On peut aussi planifier une tâche après un certain délai :

```
echo "touch nouveauFichier" | at now +5 minutes
```

- Si l'heure spécifiée est inférieure à l'heure courante, la commande est exécutée le lendemain

cron

Daemon permettant de planifier l'exécution automatique de tâches récurrentes à une date et heure données

Chaque événement est renseigné par une ligne dans une table appelée *crontab* :

minute	heure	jour du mois	mois	jour de la semaine	commande

Exemples :

`0 0 13 1 5 tâche` : *tâche* exécutée tous les 13 janvier ET tous les vendredis

`5 3 * * * apt update` : mise à jour des dépôts tous les jours à 3h05

On peut utiliser des abréviations comme `thu` pour "jeudi", ou des raccourcis comme `@monthly`, `*/5` (= toutes les 5 unités), `10-20/3` (= `10,13,16,19`)

Afficher une crontab :

```
crontab -l [-u user]
```

Editer une crontab :

```
crontab -e [-u user]
```

Supprimer une crontab :

```
crontab -r [-u user]
```

Remarques

- utilisateur par défaut = utilisateur courant
- toujours éditer ce fichier en passant par la commande `crontab -e`
- les crontab sont dans le fichier `/var/spool/cron/crontabs/`*user*[1]
- les commandes d'un fichier crontab sont exécutées avec les niveaux de permission de son propriétaire

1. Peut varier selon les distributions

Crontabs système : fichiers `/etc/crontab` et `/etc/crond.d/*`
Syntaxe légèrement différente : un champ supplémentaire (utilisateur qui exécute la commande) :

```
$cat /etc/crontab
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user command
17 * * * * root cd / && run-parts --report /etc/cron.hourly
25 6 * * * root test -x /usr/sbin/anacron || ( cd / && run-parts
--report /etc/cron.daily )
47 6 * * 7 root test -x /usr/sbin/anacron || ( cd / && run-parts
--report /etc/cron.weekly )
52 6 1 * * root test -x /usr/sbin/anacron || ( cd / && run-parts
--report /etc/cron.monthly )
```

1. `run-parts` est un script qui prend en paramètre un répertoire et exécute tous les programmes dans ce répertoire

`anacron` peut être utilisé sur des machines qui ne tournent pas 24h/24

Ce n'est pas un daemon : il vérifie grâce à des fichiers d'horodatage s'il y a des tâches à exécuter, les exécute éventuellement, puis se termine.

Autrement dit, il doit être lancé manuellement, ou par un script de démarrage, ou même par une tâche cron

`fcron` est un nouvel outil destiné à unifier et pallier les défauts du couple `cron` / `anacron`

Archivage / backup

Plan de sauvegarde

Sauvegarde = travail important du sysadmin

En cas de gros problème, on passe généralement par une restauration du système depuis une sauvegarde, ou une image du système lorsque celui-ci était encore intègre

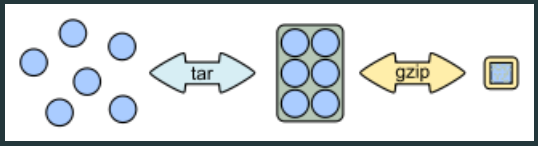
Important de définir un plan de sauvegarde et de se poser les bonnes questions :

- que faut-il sauvegarder ?
- à quelle fréquence ?
- combien de temps conservera-t-on les sauvegardes, à quel endroit, en combien d'exemplaires ?
- à quel endroit sera stocké l'*historique* des sauvegardes ?
- quel est le support le plus approprié ?
- quels sont les besoins, en capacité, du support de sauvegarde ?
- la sauvegarde doit-elle être automatique ou manuelle ?
- quelle est la méthode de sauvegarde la plus appropriée ?

tar

Outil de manipulation d'archives (*tape archiver*)

- tar concatène plusieurs fichiers en une seule archive ; on peut l'associer à un programme de *compression* :



- tar préserve l'arborescence, les droits, le propriétaire et le groupe des fichiers et des répertoires, les liens symboliques...

```
#création
$ tar cvf archive.tar fichier*
fichier1
fichier2

#listage du contenu
$ tar tvf archive.tar
-rw-rw-r-- greg/greg        0 2019-03-25 14:55 fichier1
-rw-rw-r-- greg/greg        0 2019-03-25 14:55 fichier2

#extraction
$ tar xvf archive.tar
fichier1
fichier2

#utiliser z pour compresser avec gzip
$ tar czf archive.tar.gz fichier*
$ tar xzf archive.tar.gz
```

Problème avec tar

`tar` (et `cp`) copient une arborescence ⇒ impossible de reproduire des zones du disques dur fondamentales, mais qui ne font pas partie du système de fichier (secteur de démarrage, table des partitions…)

Pour cela, on utilise la commande `dd` (*device to device*) :

```
dd if=<source> of=<cible> bs=<taille des blocs> skip= seek=
```

Attention, ne pas inverser `source` et `cible` ! Dans ce cas, `dd` devient destructeur de données !

Exemples d'utilisation de dd :

- copier un disque entier

```
dd if=/dev/sda of=/dev/sdb conv=notrunc status=progress
```

- copier seulement le MBR (secteur de démarrage) d'un disque

```
dd if=/dev/sda of=/home/$USER/MBR.image bs=446 count=1
```

- effacer un disque dur

```
dd if=/dev/zero of=/dev/sda conv=notrunc
```

- la version "parano"

```
for n in `seq 7`;
 do dd if=/dev/urandom of=/dev/sda bs=8b conv=notrunc;
done
```

- générer un gros fichier aléatoire

```
dd if=/dev/urandom of=toto bs=1M count=1024
```

- mettre le contenu d'un fichier en majuscules

```
dd if=fichier of=fichier_maj conv=ucase
```

Exemples d'utilisation de dd :

- copier un disque entier

```
dd if=/dev/sda of=/dev/sdb conv=notrunc status=progress
```

- copier seulement le MBR (secteur de démarrage) d'un disque

```
dd if=/dev/sda of=/home/$USER/MBR.image bs=446 count=1
```

- effacer un disque dur

```
dd if=/dev/zero of=/dev/sda conv=notrunc
```

- la version "parano"

```
for n in `seq 7`;
 do dd if=/dev/urandom of=/dev/sda bs=8b conv=notrunc;
done
```

- générer un gros fichier aléatoire

```
dd if=/dev/urandom of=toto bs=1M count=1024
```

- mettre le contenu d'un fichier en majuscules

```
dd if=fichier of=fichier_maj conv=ucase
```

Exemples d'utilisation de dd :

- copier un disque entier

```
dd if=/dev/sda of=/dev/sdb conv=notrunc status=progress
```

- copier seulement le MBR (secteur de démarrage) d'un disque

```
dd if=/dev/sda of=/home/$USER/MBR.image bs=446 count=1
```

- effacer un disque dur

```
dd if=/dev/zero of=/dev/sda conv=notrunc
```

- la version "parano"

```
for n in `seq 7`;
 do dd if=/dev/urandom of=/dev/sda bs=8b conv=notrunc;
done
```

- générer un gros fichier aléatoire

```
dd if=/dev/urandom of=toto bs=1M count=1024
```

- mettre le contenu d'un fichier en majuscules

```
dd if=fichier of=fichier_maj conv=ucase
```

Exemples d'utilisation de dd :

- copier un disque entier

```
dd if=/dev/sda of=/dev/sdb conv=notrunc status=progress
```

- copier seulement le MBR (secteur de démarrage) d'un disque

```
dd if=/dev/sda of=/home/$USER/MBR.image bs=446 count=1
```

- effacer un disque dur

```
dd if=/dev/zero of=/dev/sda conv=notrunc
```

- la version "parano"

```
for n in `seq 7`;
  do dd if=/dev/urandom of=/dev/sda bs=8b conv=notrunc;
done
```

- générer un gros fichier aléatoire

```
dd if=/dev/urandom of=toto bs=1M count=1024
```

- mettre le contenu d'un fichier en majuscules

```
dd if=fichier of=fichier_maj conv=ucase
```

Exemples d'utilisation de dd :

- copier un disque entier

```
dd if=/dev/sda of=/dev/sdb conv=notrunc status=progress
```

- copier seulement le MBR (secteur de démarrage) d'un disque

```
dd if=/dev/sda of=/home/$USER/MBR.image bs=446 count=1
```

- effacer un disque dur

```
dd if=/dev/zero of=/dev/sda conv=notrunc
```

- la version "parano"

```
for n in `seq 7`;
  do dd if=/dev/urandom of=/dev/sda bs=8b conv=notrunc;
done
```

- générer un gros fichier aléatoire

```
dd if=/dev/urandom of=toto bs=1M count=1024
```

- mettre le contenu d'un fichier en majuscules

```
dd if=fichier of=fichier_maj conv=ucase
```

Exemples d'utilisation de dd :

- copier un disque entier

```
dd if=/dev/sda of=/dev/sdb conv=notrunc status=progress
```

- copier seulement le MBR (secteur de démarrage) d'un disque

```
dd if=/dev/sda of=/home/$USER/MBR.image bs=446 count=1
```

- effacer un disque dur

```
dd if=/dev/zero of=/dev/sda conv=notrunc
```

- la version "parano"

```
for n in `seq 7`;
  do dd if=/dev/urandom of=/dev/sda bs=8b conv=notrunc;
done
```

- générer un gros fichier aléatoire

```
dd if=/dev/urandom of=toto bs=1M count=1024
```

- mettre le contenu d'un fichier en majuscules

```
dd if=fichier of=fichier_maj conv=ucase
```