10/2012

SYSTEMD VAINQUEUR DE UPSTART ET DES SCRIPTS « SYSTEM V » ?

par Jérôme Delamarche
[Consultant indépendant en technologies open source]

Après plus de 30 ans de service (!), le vénérable processus init et ses scripts de service disparaissent petit à petit des systèmes Unix et de nos distributions Linux préférées. Si la technologie « Upstart » est connue des adeptes d'Ubuntu, quel sera le remplaçant sur RedHat, Debian et les autres ? systemd très probablement... L'objectif de cet article est de vous préparer à la migration vers cette nouvelle technologie.

1 Introduction

Vous connaissez certainement les scripts dits « System V » ou « de service » ? Vous savez, ce sont ces scripts shell, situés sous `/etc/init.d/` et dont l'appel avec les paramètres `start` ou `stop` produit les messages : `Démarrage de .... [OK]` ou `Arrêt de ... [OK]`.

Ces scripts, ainsi que leur mécanique de lancement via `/etc/rc`, sont un héritage du vénérable Unix System V qui date de 1983. Cette architecture est simple, mais certains la trouvent lente, peu robuste et limitée, aussi depuis quelques années deux alternatives ont émergé au sein des principales distributions Linux. D'un côté, on trouve les partisans de `Upstart` (Ubuntu) et de l'autre les supporters de `systemd` (Fedora, Mandriva, OpenSuSE et bientôt RedHat et Debian). Notons que l'auteur de `systemd` – Lennart Poettering **[LPOE1]** – est employé chez RedHat... ceci explique cela.

Il ne s'agit pas ici de comparer ces deux successeurs du vénérable `init`, mais d'expliquer le fonctionnement de `systemd`. Pourquoi ce choix partial ? Parce que `systemd` me semble mieux documenté, mieux architecturé et surtout plus universel au vu du nombre de distributions qui l'ont choisi ou qui vont l'adopter.

Pourquoi systemd est-il meilleur que init ? (voir **[LPOE2]**)

Dans un souci de rationalisation et d'optimisation, `systemd` va réaliser des tâches qui étaient auparavant laissées au bon-vouloir du développeur des scripts System V. Par exemple, dorénavant, la gestion et le contrôle des PID des processus de service ainsi que ceux de leurs enfants sont à la charge de `systemd`. De même pour les logs. `Systemd` va aussi jouer le rôle de coordinateur et d'arbitre pour gérer dépendances et conflits.

Pour améliorer les performances du démarrage, `systemd` adopte plusieurs techniques astucieuses : tout d'abord, connaissant l'arbre de dépendances des services, il est naturellement capable de paralléliser les lancements de processus (on peut lancer A et B s'ils ne dépendent pas l'un de l'autre, même indirectement). Mais au-delà, `systemd` peut lancer en parallèle des processus interdépendants ! Comment cela ? `Systemd` va anticiper et créer une *socket* Unix pour tout service à démarrer. Ainsi, les « clients » d'un service (c'est à dire les services dépendants) vont se connecter dessus alors que le service réel n'est peut-être pas encore actif. Quand celui-ci le sera, `systemd` lui passera la socket, ainsi les « clients » bloqués pourront être servis !

2 Concepts et architecture de systemd

Le composant de base géré par `systemd` s'appelle une « unité » (*unit*). Il faut imaginer chaque unité comme un objet à part entière, possédant

généralement sa propre configuration. Les « anciens » scripts de service peuvent être réécrits sous forme d'unités pour `systemd`, mais ce ne seront pas les seuls !

2.1 Types d'unités

Les unités sont des objets typés qui possèdent un état, par exemple « actif », « inactif » ou « en erreur » (*active, inactive, failed*).

Les types d'unités supportés sont :

Nom du type d'unité	Description
`service`	permet de déclencher et superviser des processus lancés à travers des scripts System V ou directement par `systemd`.
`socket`	de type Unix, FIFO ou INET, elle est associée à un service. Par exemple, un accès à la socket `nscd.socket` va démarrer le service `nscd.service` (voir plus haut le principe de l'anticipation par création de sockets Unix).
`mount`	gestion du fichier `/etc/fstab`.
`automount`	pour la gestion des systèmes de fichiers auto-montés.
`path`	permet d'activer d'autres services quand des fichiers ou des répertoires sont modifiés. Éventuellement les répertoires peuvent être créés par `systemd`.
`target`	pour regrouper des unités logiquement ou pour définir des unités prédéfinies dites « unités spéciales » (voir ci-dessous).
`timer`	pour activer des unités sur le déclenchement de *timers*.
`snapshot`	pour permettre un *save/rollback* pendant des changements de configuration de service.
`swap`	pour l'activation des zones de *swap* (pas utilisé sur Fedora 17).

Nous allons voir que chaque unité possède son propre fichier de configuration, mais le type de l'unité sera donné par le suffixe du nom du fichier de configuration. Par exemple, le fichier `sshd.service` sera associé à une unité de type `service`.

2.2 Unités prédéfinies

Mais pour pouvoir se substituer au vieil `init`, il a fallu concevoir des unités prédéfinies (ou unités *spéciales*), certaines correspondant directement à la valeur du champ `action` des lignes du fichier `/etc/inittab` :

(tableau non exhaustif)

Nom de l'unité prédéfinie	Description
`basic.target`	définit des tâches à déclencher très tôt.
`ctrl-alt-del.target`	que faire quand l'utilisateur appuie sur Control-Alt-Suppr (lien symbolique vers `reboot.target`).
`default.target`	lien symbolique vers `multi-user.target` (équivalent au *runlevel 3* de RedHat/Fedora) ou `graphical.target` (équivalent au runlevel *5*). C'est l'unité déclenchée en premier.
`emergency.target`, `rescue.target` `halt.target`, `poweroff.target`, `reboot.target`, `shutdown.target`	unités invoquées au démarrage ou lors de l'arrêt du système ou quand l'ordre correspondant est donné par la commande `systemctl` (voir § 4).
`sysinit.target`	unités invoquées très tôt lors du démarrage et indépendamment du runlevel. L'activation du LVM en est un exemple.
`runlevelX.target`	(où X prend les valeurs 0 à 6) permet de simuler les changements de runlevel. Ces unités sont en fait des liens symboliques vers `poweroff.target` (niveau 0), `rescue.target` (niveau 1), `multi-user.target` (niveaux 2,3,4), `graphical.target` (niveau 5) et `reboot.target` (niveau 6).
`local-fs.target`, `remote-fs.target`, `swap.target`	pour le montage des systèmes de fichiers et l'activation du swap.
`network.target`	pour gérer correctement les dépendances entre les services et le réseau.

2.3 Organisation et syntaxe des fichiers de configuration

Si on conçoit bien le besoin d'imiter le comportement de `init`, quoique le nombre des unités spéciales soit très conséquent, il faut maintenant se représenter concrétement comment sont conçues ses fameuses unités.

`Systemd` est composé de nombreux exécutables situés sous `/usr/lib/systemd/`. D'ailleurs le premier processus, lancé par le noyau au démarrage, est `/usr/bin/systemd` qui n'est autre qu'un lien symbolique vers `/usr/lib/systemd/systemd`. Si vous ne vous en doutiez pas déjà, vous noterez dès à présent que l'architecture de `systemd` fait un usage abondant des liens symboliques...

La configuration est située dans le répertoire `/etc/systemd/` et le fichier de configuration principal va s'appeler `/etc/systemd/system.conf`.

Pour connaître l'emplacement des fichiers de configuration de **systemd**, il est possible d'utiliser la commande suivante :

```
# pkg-config systemd --variable=systemdsystemconfdir
/etc/systemd/system
```

Notes

Note 1 : **systemd** peut être invoqué par un utilisateur avec l'option **--user**, dans ce cas, son fichier de configuration principal est **/etc/systemd/user.conf**, mais nous occulterons cette possibilité ici.

Note 2 : Sur la distribution Fedora 17, utilisée ici comme exemple, les répertoires **/bin**, **/sbin** et **/lib** sont en fait des liens symboliques sur **/usr/bin**, **/usr/sbin** et **/usr/lib**.

Le fichier **/etc/systemd/system.conf** est un fichier « à la .INI », contenant une rubrique **[Manager]** et des options globales, dont les principales sont listées dans ce tableau :

Paramètre	Description	Valeur par défaut
LogLevel	définit le niveau de verbosité.	info
LogTarget	destination des messages de logs. Une valeur usuelle est **journal**, qui exploite un processus supplémentaire (voir §3.2).	**syslog**
MountAuto	honore les montages de systèmes de fichiers listés dans **/etc/fstab**.	yes
MountSwap	honore l'activation des zones de swap listées dans **/etc/fstab**.	yes
Default-Controllers	choix du *cgroup*.	cpu

Certaines options peuvent être surchargées en passant des options au noyau ou sur la ligne de commandes **systemd**. Par exemple, en cas de problème, **systemd** propose un mode « pas-à-pas » pour permettre la résolution des problèmes si vous lui passez le paramètre **systemd.confirm_spawn=1** comme option du noyau, ou **--confirm-spawn** sur sa ligne de commandes.

systemd analyse ensuite le contenu du fichier bien nommé **/etc/systemd/system/default.target** qui, sur la Fedora 17 lancée en mode graphique, est un lien symbolique vers **/lib/systemd/system/runlevel5.target**. Enfin du concret ! Nous pouvons examiner le contenu de ce fichier et commencer à explorer la syntaxe des fichiers de configuration des unités.

Notez que sur votre distribution, pour connaître l'emplacement des fichiers de configuration des unités, il est possible d'utiliser la commande suivante :

```
# pkg-config systemd --variable=systemdsystemunitdir
/usr/lib/systemd/system
```

Si les fichiers qui sont sous **/etc/systemd/system/** sont en fait des liens symboliques sur leurs homonymes situés sous **/lib/systemd/system/**, certains répertoires existent avec le même nom à la fois sous les répertoires **/etc/systemd/system/** **ET** **/lib/systemd/system/**, et dans ce cas leur contenu diffère. L'affichage de la valeur de la variable **systemdsystemunitpath** avec la commande **pkg-config** montre bien que le répertoire **/etc/systemd/system/** est prioritaire.

Un fichier de configuration d'unité contient les rubriques suivantes :

Nom de la rubrique	Description
[Unit]	décrit le service et les règles de dépendances à l'aide des paramètres : *Description, Before, After, Requires, Requisite, Wants, Condition**.
[Install]	décrit ce qu'il faut faire quand on invoque **systemctl enable | disable**. Par exemple, le paramètre **WantedBy=xxxx** indique qu'il faudra créer un lien symbolique dans le répertoire **/etc/systemd/system/xxxx.wants/**. **Also** indique des unités additionnelles à installer en plus.

Puis, selon le type de l'unité, des sections supplémentaires et leurs lots de paramètres seront nécessaires.

2.4 Exemples de configuration

À tout seigneur tout honneur, examinons le contenu du fichier **/etc/systemd/system/default.target** qui est le nom de la première unité lancée par **systemd** (pour information, je suis un adepte de la commande **readlink** qui permet de connaître le nom final du fichier référencé par une suite de liens symboliques) :

```
# readlink -f /etc/systemd/system/default.target
/lib/systemd/system/graphical.target
# cat /etc/systemd/system/default.target
```

```
[Unit]
Description=Graphical Interface
Documentation=man:systemd.special(7)
Requires=multi-user.target
After=multi-user.target
Conflicts=rescue.target
AllowIsolate=yes

[Install]
Alias=default.target
```

Nous retrouvons bien les sections **[Unit]** et **[Install]** dont nous avons parlé au §2.3. Le paramètre **Requires=** indique que l'unité **multi-user.target** doit être activée pour

que `default.target` soit considérée comme active. Le paramètre `After=` précise qu'il faut attendre la fin de l'exécution de l'unité `multi-user.target` avant de finir l'exécution de `default.target` (il existe aussi un paramètre `Before=`, mais en l'absence de précision sur l'ordonnancement, les exécutions sont parallélisées).

Le paramètre `Conflicts=` indique que l'unité `default.target` ne doit pas être lancée si `rescue.target` est déjà activée. L'unité `rescue.target` correspond au runlevel 1, il est normal de ne pas lancer le mode multi-utilisateur et encore moins l'interface graphique !

Le paramètre `AllowIsolate=` est un *hack* : il permettra de faire en sorte qu'on puisse imiter le changement de niveau d'exécution comme on le faisait avec `init`. Dans le cas d'un changement de runlevel, donc, tous les processus qui ne seront pas gérés par l'unité `default.target` ou ses dépendances seront arrêtés.

Le paramètre `Alias=` de la rubrique `[Install]` indique que lors de l'activation de cette unité (dont le véritable nom est `runlevel5.target`), il faudra créer un alias sous la forme d'un lien symbolique nommé `default.target` ! Eh oui : autant décrire tous les liens symboliques dans des fichiers de configuration et laisser les outils fournis avec `systemd` gérer correctement ces liens.

Reprenons : `default.target` doit être activée après `multi-user.target`. Que fait cette unité ?

```
# cat /lib/systemd/system/multi-user.target
```

```
[Unit]
Description=Muti-User
Documentation=man:systemd.special(7)
Requires=basic.target
Conflicts=rescue.service rescue.target
After=basic.target rescue.service rescue.target
AllowIsolate=yes

[Install]
Alias=default.target
```

Vous commencez à comprendre le mécanisme ? On nage en pleine récursivité : cette unité dépend de `basic.target`, mais à un moment donné, il faudra bien lancer des processus ! Notons que la rubrique `[Install]` définit un Alias qui porte le même nom que celui défini par l'unité `graphical.target` : ce n'est pas un problème car l'unité `graphical.target` se jouant après `multi-user.target`, le lien symbolique `default.target` pointera bien vers la bonne unité.

En continuant l'examen des unités selon leurs dépendances, on constate que `basic.target` dépend de `sysinit.target` et `sockets.target`. La configuration de `sysinit.target` contient un nouveau paramètre intéressant :

```
[Unit]
...
Wants=local-fs.target swap.target
```

On se doute qu'il va falloir d'abord activer les systèmes de fichiers locaux ainsi que le *swap* avant de réaliser d'autres tâches, mais ici, `Wants=` indique que l'unité `sysinit.target` sera activée même si l'une des unités dont elle dépend échoue lors de son activation. Ce comportement est différent du `Requires=` qui stipule que l'unité n'est activée que si toutes les unités dont elle dépend sont activées avec succès (c'est une forme de transaction !).

Parfois, une unité peut dépendre de très nombreuses autres unités, ce qui provoquerait l'écriture de directives `Wants=` ou `Requires=` assez fastidieuses avec le souci de devoir modifier ces lignes chaque fois que les dépendances sont modifiées. Pour éviter cela, `systemd` permet d'externaliser les dépendances d'une unité en permettant l'utilisation de répertoires associés à des liens symboliques (encore !). Le principe est le suivant (nous le décrivons pour la directive `Wants=`, il se transpose aisément pour la directive `Requires=`) :

Prenons l'exemple de l'unité `sysinit.target`, elle dépend de `local-fs.target` et `swap.target`, comme indiqué par son fichier de configuration, mais ce n'est pas tout ! Comme il existe un répertoire nommé `/etc/systemd/system/sysinit.target.wants/`, l'unité `sysinit.target` dépend aussi de toutes les unités dont le nom apparaît dans les liens symboliques situés dans ce répertoire :

```
# cd /etc/systemd/system
# ls -l sysinit.target.wants
...  lvm2-monitor.service -> /usr/lib/systemd/system/lvm2-monitor.service
...  mdmonitor-takeover.service -> /usr/lib/systemd/system/mdmonitor-takeover.service
```

Les dépendances se règlent donc simplement à coups de création et destruction de liens symboliques. Rôle parfaitement rempli par la commande `systemctl` que nous verrons dans un instant.

Prenons un autre exemple : comment est lancé le démon `sshd` ? Il est décrit par l'unité `/lib/systemd/system/sshd.service` qui est référencée par un lien symbolique situé sous `/etc/systemd/system/multi-user.target.wants/`. Le contenu du fichier `sshd.service` est :

```
[Unit]
Description=OpenSSH server saemon
After=syslog.target network.target auditd.service

[Service]
EnvironmentFile=/etc/sysconfig/sshd
ExecStartPre=/usr/sbin/sshd-keygen
ExecStart=/usr/sbin/sshd -D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID

[Install]
WantedBy=multi-user.target
```

Oui, un service dispose d'une nouvelle rubrique nommée [Service] dont les paramètres sont explicites. Toutefois, il manque ici le paramètre Type= qui indique comment lancer le service. La valeur par défaut est simple pour indiquer que le processus nommé par ExecStart= est le processus principal du service, mais les autres possibilités sont : forking, dbus, oneshot, idle et notify (voir la documentation man systemd.service). Remarquez aussi la ligne de commandes qui permet de faire un « reload » du démon : la variable $MAINPID est gérée par systemd et aura pour valeur le PID de sshd.

Dans la configuration de l'unité sshd.service, on trouve aussi la directive WantedBy= qui indique la dépendance entre ce service et l'unité multi-user.target, ce qui nécessitera la création d'un lien symbolique dans /etc/systemd/system/multi-user.target.wants/ lors de l'initialisation des unités.

La configuration des services permet aussi d'imiter la fonctionnalité du respawn proposée par init. Avec systemd, la surveillance et la regénération d'un processus de service est réalisée par les directives Restart= (sur quel état du processus faut-il le relancer ?) et RestartSec= (au bout de combien de secondes faut-il le relancer ?). Regardez le contenu de la configuration de l'unité getty@.service pour en avoir un exemple.

Avez-vous remarqué la présence du caractère @ dans le nom de l'unité getty@.service ? Il s'agit d'une astuce lexicale proposée par systemd pour permettre de créer des modèles d'unités (*unit templates*). En effet, vous savez qu'habituellement vous disposez d'un certain nombre de terminaux virtuels, chacun étant géré par une commande de la famille getty (par exemple, sur RedHat cette commande est /sbin/mingetty – à voir dans le fichier /etc/inittab). Pour imiter ce comportement, il faudrait donc définir autant d'unités que de terminaux virtuels à gérer, ce qui est évidemment à éviter. Les modèles d'unités prennent tout leur sens alors :

si systemd doit activer une unité dont le nom est getty@tty1.service et que ce nom d'unité n'existe pas sous /lib/systemd/system/, alors, à cause de la présence du caractère @ dans le nom de l'unité, systemd va rechercher une unité dont le nom est getty@.service. Comme cette unité existe, c'est elle qui sera invoquée et la chaîne tty1 qui faisait partie du nom de l'unité initiale sera fournie sous forme de « specifier » accessible dans la configuration de l'unité. Ce *specifier* se nomme %I (lire « i » majuscule. Il en existe d'autres – voir man systemd.unit).

Voici un extrait du fichier getty@.service sur Fedora17 :

```
[Unit]
Description=Getty on %I
...
[Service]
Restart=always
RestartSec=0
TTYPath=/dev/%I
...
```

Outre les services, au sens « classique » du terme, nous avons dit en préambule que systemd allait aussi gérer les sockets afin d'accélérer les lancements des processus inter-dépendants. Regardons la syntaxe de configuration d'une unité de type Socket, par exemple cups.socket :

```
[Unit]
Description=CUPS Printing Service Sockets

[Socket]
ListenStream=/var/run/cups/cups.sock
ListenStream=631
ListenDatagram=0.0.0.0:631

[Install]
WantedBy=sockets.target
```

Nous constatons la présence d'une rubrique [Socket] contenant les paramètres de création des sockets à créer. Ici, systemd va créer une socket du domaine UNIX qui s'appellera /var/run/cups/cups.sock, puis une socket TCP (ListenStream) à l'écoute sur le port 631 pour toutes les adresses IP de la machine, puis il créera une socket UDP (ListenDatagram) à l'écoute du port 631 en IPv4 uniquement. D'autres paramètres sont disponibles (voir man systemd.socket).

Comme ultime exemple, nous allons prendre l'exemple d'un montage à effectuer lors du démarrage. Il m'arrive souvent de placer ce type de montage dans le vénérable /etc/rc.local, mais le faire gérer par systemd est un gage de robustesse. Sur notre Fedora 17, le contenu du fichier media.mount est :

```
[Unit]
Description=Media Directory
Before=local-fs.target

[Mount] :
What=tmpfs
Where=/media
Type=tmpfs
Options=mode=755,nosuid,nodev,noexec
```

Ce qui permet d'avoir au démarrage un répertoire /media associé à un système de fichiers de type tmpfs.

Comme vous le subodorez peut-être déjà, le nombre de paramètres possibles pour définir la configuration des unités est très important, ce qui fait à la fois la richesse et la complexité de systemd !

2.5 Petite synthèse

Récapitulons en prenant comme exemple le démarrage du système : voir Figure 1, page suivante.

Lorsque systemd est lancé par le noyau (1), il lit son fichier /etc/systemd/system.conf (2), puis il examine la première unité à lancer : default.target (3) qui est en réalité un lien symbolique vers runlevel5.target (lui-même lien symbolique vers graphical.target).

Ce fichier contient une directive After=multi-user.target, donc systemd cherche la configuration de cette unité (4) et l'interprète. Mais il existe un répertoire multi-user.target.wants (5), donc

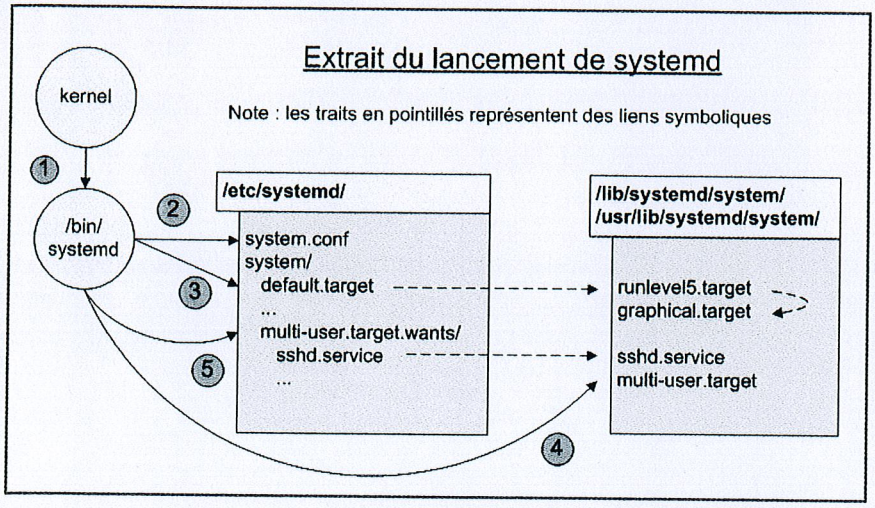

Figure 1

`systemd` parcourt ce répertoire pour activer les unités qui y sont décrites. On y trouve par exemple `sshd.service`, etc.

3 Gestion des processus et des logs

3.1 Gestion des processus

Par défaut, `systemd` maintient automatiquement la relation entre PID et processus activés. Ainsi, le PID de notre démon `sshd` précédent est stocké par `systemd` dans le fichier `/var/run/sshd.pid`. Mais pour les services dont la valeur de `Type=` est `forked` (alors que le type est `simple` pour `sshd.service`), `systemd` peut se tromper dans son analyse du processus principal. En effet, quand un service est de type `forked`, cela signifie que le processus lancé par `systemd` va lancer un ou plusieurs processus fils puis va se terminer. S'il a lancé plusieurs processus fils, le stockage de PID du processus fils « principal » est à la charge du processus parent et il faut indiquer à `systemd` dans quel fichier est stocké ce PID. On fait pour cela un usage de la directive `PIDFile=` dans le fichier de configuration de l'unité.

Note

Note : sur Fedora 17, le répertoire `/var/run` est en fait un lien symbolique vers `/run` qui est le point de montage d'un système de fichiers de type `tmpfs`, donc rapide mais non persistant.

3.2 Gestion des logs

Dans le tableau du §2.3, nous avons montré les paramètres principaux de `systemd` dans le fichier `/etc/systemd/system.conf`. Parmi ceux-ci, nous avons évoqué `LogTarget=`. Sur Fedora 17, la valeur de ce paramètre est : `journal`. Cela indique à `systemd` d'envoyer tous les messages de log à un nouveau démon nommé `systemd-journald` lui-même activé par `systemd` car décrit par l'unité `systemd-journald.service`.

Ce processus est configuré par le fichier `/etc/systemd/systemd-journald.conf` dont voici un extrait :

```
[Journal]
ForwardToSyslog=yes
MaxLevelSyslog=debug
Compress=yes
#SystemMaxUse=
#ForwardToConsole=no
```

Le processus `systemd-journald` stocke les logs qu'il reçoit dans des fichiers situés sous `/run/log/journal/`, mais il peut aussi réémettre les logs vers le démon `syslog` (ou `rsyslog` dans notre cas) ou vers la Console système. On peut lui indiquer des tailles limites de fichiers à générer ou une taille maximale d'espace disque à utiliser.

Les fichiers de logs sont à un format binaire, il faut utiliser la commande `journalctl` pour en afficher le contenu :

```
# journalctl
```

affiche le contenu des logs avec votre paginateur préféré, mais il est possible de limiter le nombre de lignes affichées avec l'option `-n` (comme le fait la commande `tail`) :

```
# journalctl -n 10
```

Enfin, l'affichage peut être filtré (comme le ferait `grep` !) en précisant des valeurs spécifiques pour les champs des lignes de logs. Ces champs ont des noms prédéfinis comme `MESSAGE`, `PRIORITY`, `_PID`, `_HOSTNAME`, `_SYSTEMD_UNIT` (voir man `systemd.journal-fields`). Par exemple, pour n'afficher que les logs relatifs à l'unité `sshd.service`, on exécute la commande :

```
# journalctl _SYSTEMD_UNIT=sshd.service
```

4 Comment faire « comme avant, avec init » ?

Finalement, 30 ans après, Unix/Linux ne change pas radicalement : on peut gérer `systemd` avec les commandes `ln` pour les liens symboliques et `vi` pour la configuration ! Néanmoins, essayons d'être un peu plus productifs et évitons de réécrire des commandes qui existent déjà. Si, comme moi, vous vous êtes habitué aux commandes simples mais néanmoins utiles `chkconfig` et `service`, par quoi va-t-il falloir les remplacer ?

Voici un petit tableau qui met en parallèle vos commandes usuelles avec leur pendant « à la mode de » `systemd` :

Remplacement de la commande `chkconfig`	
`chkconfig sshd on`	`systemctl enable sshd.service`
`chkconfig sshd off`	`systemctl disable sshd.service`
`chkconfig sshd --list`	`systemctl is-enabled sshd.service`
Remplacement de la commande `service`	
`service sshd status`	`systemctl status sshd.service` `service sshd status`
`service sshd start`	`systemctl start sshd.service` `service sshd start`
`service sshd stop`	`systemctl stop sshd.service` `service sshd stop`
`service sshd reload`	`systemctl reload sshd.service` `service sshd reload`
Remplacement des commandes `init` et `telinit`	
`init 0`	`systemctl poweroff` `init 0`
`init 6`	`systemctl reboot` `init 6`
`init 3`	`systemctl isolate runlevel3.target` `init 3`
Choix du niveau d'exécution au démarrage	
valeur de `initdefault` dans `/etc/inittab`	valeur du lien symbolique `default.target`

En résumé :

Fonctionnalité	init (system V)	systemd
Activation d'un service au démarrage	commande `chkconfig` liens symboliques vers `/etc/init.d/`	`chkconfig` et liens symboliques pour les scripts non encore migrés commande `systemctl` pour les processus pris en charge par `systemd`
Démarrage ou arrêt d'un service	commande `service`	commande `service` ou commande `systemctl`
Changement de niveau d'exécution	commande `init`	commande `init` ou commande `systemctl`

Vous voyez l'intérêt de la nouvelle commande `systemctl` qui centralise toutes les opérations mais qui permet de réaliser d'autres opérations :

- pour obtenir la liste de toutes les unités (situées sous `/lib/systemd/system/`) et leur état courant :

```
# systemctl list-unit-files
```

- pour obtenir la liste et la description des unités de type `service` :

```
systemctl list-units -t service
```

- si vous changez la configuration de `systemd` ou des unités, la commande suivante permet de faire relire les fichiers par `systemd` :

```
# systemctl daemon-reload
```

- et si vous aimez les graphes bien « touffus » (et inexploitables !), vous pouvez générer une image représentant les dépendances entre toutes les unités par :

```
# systemctl dor | dot -Tsvg > fichier.svg
```

Conclusion

À la lecture de cet article, peut-être avez-vous eu la réaction suivante : « mais pourquoi avoir fait si compliqué alors que finalement `init` ne fonctionnait pas si mal que ça ? » [NDLR : ce serait plutôt « mais c'est quoi ce #@$!& ? »]. Effectivement, je me suis aussi posé cette question et me la pose encore. Administrateur de serveurs Linux, je n'ai jamais connu de problème avec `init` et, si parfois, on peut considérer le temps de démarrage comme plutôt long, ce n'est rien comparé aux délais de scans et de tests des Bios et *firmwares* des contrôleurs de toutes sortes. Le passage vers `systemd` (ou `upstart`) ne se justifie pas, selon moi, sur de telles plates-formes. Vouloir « faire la course » entre distributions Linux, Mac OS X voire Windows (!), n'a aucun intérêt pour des serveurs. Néanmoins, il semble que le pas soit pris et que, petit à petit, les principales distributions Linux nous forcent ainsi à nous adapter. Alors, avant de vous retrouver face à une nouvelle version de votre Linux préféré où `init` et les scripts System V auront définitivement disparu, étudiez dès à présent le fonctionnement de `systemd` dont le périmètre d'action va encore s'accroître avec la future intégration de `udev` ! ■

Références

[LPOE1] Blog de l'auteur Lennart Poettering (http://0pointer.de/blog/projects/systemd.html)

[LPOE2] Comparaison faite par L.Poettering entre systemd, upstart et sysvinit (http://0pointer.de/blog/projects/why.html)

[FEDORA] le wiki de systemd sur Fedoraproject.or (http://fedoraproject.org/wiki/Systemd)