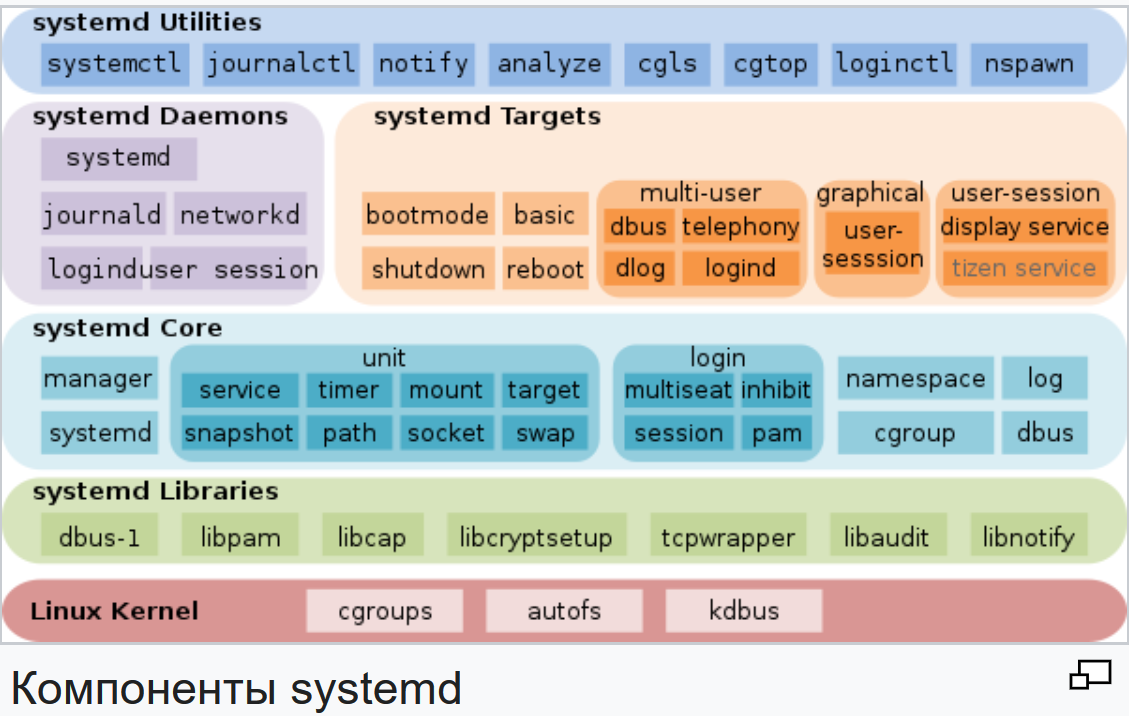
## https://ru.wikipedia.org/wiki/Systemd



https://debian-handbook.info/browse/ru-RU/stable/unix-services.html#sect.systemd

## 9.1. Загрузка системы

При загрузке компьютера множество сообщений, пробегающих на консоли, отображает выполнение многочисленных автоматических начальных инициализаций и настроек. Иногда может возникнуть желание несколько изменить работу этого этапа, а значит, необходимо хорошо её понимать. Помочь в этом — назначение данного раздела.

Сначала BIOS получает контроль над компьютером, определяет диски, считывает главную загрузочную запись и запускает загрузчик. Загрузчик принимает управление, находит ядро на диске, считывает и запускает его. Затем ядро инициализируется и начинает поиск и монтирование корневой файловой системы и, наконец, запускает первую программу — **init**. Зачастую эти «корневой раздел» и **init** на самом деле находятся на виртуальной файловой системе, существующей только в ОЗУ (отсюда её название — initramfs, ранее — initrd, от "initialization RAM disk"). Эта файловая система загружается в память загрузчиком, часто из файла на жёстком диске или по сети. Он содержит самый минимум, необходимый для того, чтобы ядро загрузило «настоящую» корневую файловую систему: сюда могут входить модуля ядра для жёсткого диска или других устройств, без которых система не способна загрузиться, или, чаще, сценарии инициализации и модули для сборки массивов RAID, открытия зашифрованных разделов, активации томов LVM и т. п. Когда корневой раздел примонтирован, initramfs передаёт управление настоящему процессу "init", и система возвращается к стандартному процессу загрузки.

### 9.1.1. Система инициализации systemd

«Настоящий init» сейчас предоставляется systemd, и в данном разделе описывается эта система инициализации.

КУЛЬТУРА**До systemd**

**systemd** является относительно новой «системой инициализации», но несмотря на это, она уже была доступна в Wheezy. По умолчанию, она стала применяться в Debian Jessie. До этого, по умолчанию использовалась «System V init» (пакет sysv-rc) — гораздо более традиционная система. System V init будет описана позднее.

АЛЬТЕРНАТИВА**Другие системы загрузки**

This book describes the boot system used by default in Debian Buster (as implemented by the systemd package), as well as the previous default, sysvinit, which is derived and inherited from System V Unix systems; there are others.

file-rc — это очень просто устроенная система загрузки. Она сохраняет принцип уровней запуска, но заменяет каталоги и символьные ссылки конфигурационным файлом, указывающим **init**, какие процессы и в каком порядке должны быть запущены.

The **upstart** system is still not perfectly tested on Debian. It is event based: init scripts are no longer executed in a sequential order but in response to events such as the completion of another script upon which they are dependent. This system, started by Ubuntu, was present in Debian Jessie, but was not the default; it came, in fact, as a replacement for sysvinit, and one of the tasks launched by **upstart** was to launch the scripts written for traditional systems, especially those from the sysv-rc package.

Есть и другие системы и другие режимы работы, такие как **runit** или **minit**, они не распространены и достаточно специализированы.

**Рисунок 9.1. Порядок загрузки компьютера с Linux и systemd**

ОСОБЫЙ СЛУЧАЙ**Загрузка по сети**

В некоторых конфигурациях BIOS может быть настроен не на загрузку MBR, а на поиск её эквивалента в сети, что делает возможным собирать компьютеры без жёсткого диска или полностью переустанавливаемые при каждой загрузке. Такая опция есть не у любого оборудования, и для неё обычно необходимо определённое сочетание BIOS и сетевой карты.

Загрузка по сети может использоваться для запуска **debian-installer** или FAI (см. [Раздел 4.1, «Способы Установки»](K:/debian-handbook/installation.html" \l "sect.installation-methods)).

К ОСНОВАМ**Процесс — экземпляр программы**

**Процесс** является отображением запущенной программы в памяти. Он содержит всю информацию, необходимую для корректной работы программы (сам код, а также данные, хранимые ей в памяти, список открытых ей файлов, установленных ей сетевых соединений и т. п.). Одна программа может быть представлена несколькими процессами, необязательно запущенными от имени разных пользователей.

БЕЗОПАСНОСТЬ**Использование командной оболочки в качестве init для получения привилегий root**

Принято, что первым процессом при загрузке является **init** (по умолчанию символьная ссылка на **/lib/systemd/systemd**). Возможно передать ядру другие параметры для **init**.

Любой человек, способный получить доступ к компьютеру, может нажать кнопку **Reset** и перезагрузить его. Потом, в приглашении загрузчика, можно передать ядру опцию **init=/bin/sh**, чтобы получить доступ root, не зная пароля администратора.

Чтобы предотвратить это, можно защитить паролем сам загрузчик. Следует также задуматься о защите доступа к BIOS (механизм защиты паролем почти всегда доступен), без которой злоумышленник всё равно сможет загрузить машину с переносного накопителя c собственной системой Linux, которую он сможет использовать для доступа к данным на жёстком диске компьютера.

Наконец, нужно знать, что в большинстве BIOS есть пароль, предназначенный для исправления неполадок и для тех, кто забыл установленный пароль. Сейчас, такие пароли находятся в свободном доступе в интернете (попробуйте найти для своего BIOS, набрав «generic BIOS passwords» в поисковике). Следует понимать, что не существует надёжного способа защитить компьютер от проникновения, если злоумышленник имеет к нему прямой способ. Например, он может просто забрать жёсткий диск, или целый компьютер, или стереть память BIOS для сброса пароля…

Systemd executes several processes, in charge of setting up the system: keyboard, drivers, filesystems, network, services. It does this while keeping a global view of the system as a whole, and the requirements of the components. Each component is described by a “unit file” (sometimes more); the general syntax is derived from the widely-used “\*.ini files“ syntax, with ***key* = *value*** pairs grouped between **[*section*]** headers. Unit files are stored under **/lib/systemd/system/** and **/etc/systemd/system/**; they come in several flavors, but we will focus on “services” and “targets” here.

Файл service описывает процесс systemd. Он содержит ту же информацию, что и прежние init-сценарии, но она описана в декларативном (и сжатом) стиле. Systemd обслуживает повторяющиеся задачи (старт и остановка процесса, проверка его статуса, журнал, привилегии и т . п.) и файл service должен быть наполнен конкретикой, относящейся к процессу. Например для SSH, он выглядит так:

[Unit]

Description=OpenBSD Secure Shell server

After=network.target auditd.service

ConditionPathExists=!/etc/ssh/sshd\_not\_to\_be\_run

[Service]

EnvironmentFile=-/etc/default/ssh

ExecStart=/usr/sbin/sshd -D $SSHD\_OPTS

ExecReload=/bin/kill -HUP $MAINPID

KillMode=process

Restart=on-failure

[Install]

WantedBy=multi-user.target

Alias=sshd.service

Как видно из примера, кода очень мало, только объявления. Также systemd описывает ход процесса, отслеживает их выполнение и даже перезапускает, если необходимо.

Файл «цели» в systemd описывает состояние системы, когда функционирует некоторый набор сервисов. Его можно рассматривать как эквивалент уровня запуска. Одна из целей — **local-fs.target**; при её достижении остальная система может рассчитывать, что все локальные файловые системы смонтированы и доступны. В число других целей входят **network-online.target** и **sound.target**. Зависимости цели могут быть указаны как внутри файла target (в строке **Requires=**), так и с использованием символьной ссылки на файл service в каталоге **/lib/systemd/system/*targetname*.target.wants/**. Например **/etc/systemd/system/printer.target.wants/** cодержит ссылку на **/lib/systemd/system/cups.service**, поэтому systemd запустит CUPS для достижения цели **printer.target**.

Так как файлы unit декларативны, в отличие от сценарием и программ, они не могут запускаться отдельно и интерпретируются только systemd, хотя несмотря на это, несколько вспомогательных программ позволяют администратору взаимодействовать с systemd, контролировать состояние системы и отдельных компонентов.

Первая из них — **systemctl**. При запуске без параметров, выводится список всех unit-файлов, известных системе (за исключением отключенных) и их статус. **systemctl status** дает лучший обзор сервисов и связанных процессов. Выводится имя файла service (как в **systemctl status ntp.service**), также дополнительная информация и последние несколько строчек из журнала, касающиеся этого процесса (позднее про это будет сказано более подробно).

Для запуска сервиса вручную, нужно просто набрать **systemctl start *servicename*.service**. Как можно догадаться, для остановки: **systemctl stop *servicename*.service**. Есть другие подкоманды: **reload** и **restart**.

Для контроля за активность сервиса (запускается при загрузки системы или нет), нужно использовать **systemctl enable *servicename*.service** (или **disable**). Для проверки запущен ли сервис — **is-enabled**.

An interesting feature of systemd is that it includes a logging component named **journald**. It comes as a complement to more traditional logging systems such as **syslogd**, but it adds interesting features such as a formal link between a service and the messages it generates, and the ability to capture error messages generated by its initialization sequence. The messages can be displayed later on, with a little help from the **journalctl** command. Without any arguments, it simply spews all log messages that occurred since system boot; it will rarely be used in such a manner. Most of the time, it will be used with a service identifier:

# **journalctl -u ssh.service**-- Logs begin at Tue 2015-03-31 10:08:49 CEST, end at Tue 2015-03-31 17:06:02 CEST. --

Mar 31 10:08:55 mirtuel sshd[430]: Server listening on 0.0.0.0 port 22.

Mar 31 10:08:55 mirtuel sshd[430]: Server listening on :: port 22.

Mar 31 10:09:00 mirtuel sshd[430]: Received SIGHUP; restarting.

Mar 31 10:09:00 mirtuel sshd[430]: Server listening on 0.0.0.0 port 22.

Mar 31 10:09:00 mirtuel sshd[430]: Server listening on :: port 22.

Mar 31 10:09:32 mirtuel sshd[1151]: Accepted password for roland from 192.168.1.129 port 53394 ssh2

Mar 31 10:09:32 mirtuel sshd[1151]: pam\_unix(sshd:session): session opened for user roland by (uid=0)

Другой полезный флаг **-f** используется с **journalctl** для просмотра появления новых сообщений (похоже на **tail -f *file***).

Если сервис не работает как ожидалось, то первым делом нужно проверить его статус с **systemctl status**, если проблема не решена, то проверьте его журнал. Допустим сервер SSH не работает:

# **systemctl status ssh.service**● ssh.service - OpenBSD Secure Shell server

Loaded: loaded (/lib/systemd/system/ssh.service; enabled)

Active: failed (Result: start-limit) since Tue 2015-03-31 17:30:36 CEST; 1s ago

Process: 1023 ExecReload=/bin/kill -HUP $MAINPID (code=exited, status=0/SUCCESS)

Process: 1188 ExecStart=/usr/sbin/sshd -D $SSHD\_OPTS (code=exited, status=255)

Main PID: 1188 (code=exited, status=255)

Mar 31 17:30:36 mirtuel systemd[1]: ssh.service: main process exited, code=exited, status=255/n/a

Mar 31 17:30:36 mirtuel systemd[1]: Unit ssh.service entered failed state.

Mar 31 17:30:36 mirtuel systemd[1]: ssh.service start request repeated too quickly, refusing to start.

Mar 31 17:30:36 mirtuel systemd[1]: Failed to start OpenBSD Secure Shell server.

Mar 31 17:30:36 mirtuel systemd[1]: Unit ssh.service entered failed state.

# **journalctl -u ssh.service**-- Logs begin at Tue 2015-03-31 17:29:27 CEST, end at Tue 2015-03-31 17:30:36 CEST. --

Mar 31 17:29:27 mirtuel sshd[424]: Server listening on 0.0.0.0 port 22.

Mar 31 17:29:27 mirtuel sshd[424]: Server listening on :: port 22.

Mar 31 17:29:29 mirtuel sshd[424]: Received SIGHUP; restarting.

Mar 31 17:29:29 mirtuel sshd[424]: Server listening on 0.0.0.0 port 22.

Mar 31 17:29:29 mirtuel sshd[424]: Server listening on :: port 22.

Mar 31 17:30:10 mirtuel sshd[1147]: Accepted password for roland from 192.168.1.129 port 38742 ssh2

Mar 31 17:30:10 mirtuel sshd[1147]: pam\_unix(sshd:session): session opened for user roland by (uid=0)

Mar 31 17:30:35 mirtuel sshd[1180]: /etc/ssh/sshd\_config line 28: unsupported option "yess".

Mar 31 17:30:35 mirtuel systemd[1]: ssh.service: main process exited, code=exited, status=255/n/a

Mar 31 17:30:35 mirtuel systemd[1]: Unit ssh.service entered failed state.

Mar 31 17:30:35 mirtuel sshd[1182]: /etc/ssh/sshd\_config line 28: unsupported option "yess".

Mar 31 17:30:35 mirtuel systemd[1]: ssh.service: main process exited, code=exited, status=255/n/a

Mar 31 17:30:35 mirtuel systemd[1]: Unit ssh.service entered failed state.

Mar 31 17:30:35 mirtuel sshd[1184]: /etc/ssh/sshd\_config line 28: unsupported option "yess".

Mar 31 17:30:35 mirtuel systemd[1]: ssh.service: main process exited, code=exited, status=255/n/a

Mar 31 17:30:35 mirtuel systemd[1]: Unit ssh.service entered failed state.

Mar 31 17:30:36 mirtuel sshd[1186]: /etc/ssh/sshd\_config line 28: unsupported option "yess".

Mar 31 17:30:36 mirtuel systemd[1]: ssh.service: main process exited, code=exited, status=255/n/a

Mar 31 17:30:36 mirtuel systemd[1]: Unit ssh.service entered failed state.

Mar 31 17:30:36 mirtuel sshd[1188]: /etc/ssh/sshd\_config line 28: unsupported option "yess".

Mar 31 17:30:36 mirtuel systemd[1]: ssh.service: main process exited, code=exited, status=255/n/a

Mar 31 17:30:36 mirtuel systemd[1]: Unit ssh.service entered failed state.

Mar 31 17:30:36 mirtuel systemd[1]: ssh.service start request repeated too quickly, refusing to start.

Mar 31 17:30:36 mirtuel systemd[1]: Failed to start OpenBSD Secure Shell server.

Mar 31 17:30:36 mirtuel systemd[1]: Unit ssh.service entered failed state.

# **vi /etc/ssh/sshd\_config**# **systemctl start ssh.service**# **systemctl status ssh.service**● ssh.service - OpenBSD Secure Shell server

Loaded: loaded (/lib/systemd/system/ssh.service; enabled)

Active: active (running) since Tue 2015-03-31 17:31:09 CEST; 2s ago

Process: 1023 ExecReload=/bin/kill -HUP $MAINPID (code=exited, status=0/SUCCESS)

Main PID: 1222 (sshd)

CGroup: /system.slice/ssh.service

└─1222 /usr/sbin/sshd -D

#

После проверки статуса (ошибка), был проверен журнал, была обнаружена ошибка в конфигурационном файле. После его редактирования и исправления ошибки, сервис запускается заново, далее проверяется его статус.

УГЛУБЛЯЕМСЯ**Другие типы unit файла**

Мы описали только базовые возможности systemd, но эта система предлагает много других интересных возможностей. Обозначим некоторые из них:

активация сокета: «socket» unit файл используется для описания сети или Unix сокета. Это означает, что сокет создаётся systemd и текущий сервис может запускаться по запросу, если будет попытка соединения. Тут копируется набор возможностей **inetd**. Читайте systemd.socket(5).

таймеры: «timer» unit файл описывает события, возникающие с установленной частотой или в определённый момент времени. Если сервис ссылается на таймер, процесс будет запущен в установленное время. Тут копируется часть возможностей **cron**. Читайте systemd.timer(5).

сеть: «network» unit файл описывает сетевой интерфейс и позволяет его настраивать, а также отражать зависимость сервиса от какого-либо интерфейса.