

# systemd

## Пишем systemd Unit файл

Systemd Unit — это скрипт, который выполняет различные действия ( все то, что пропишешь в него). Я обычно пишу такие скрипты для запуска различных служб которые компилирую руками или когда использую готовое ПО.

### Что дадут вам Systemd юниты?

Units — это объекты, которыми может управлять система (В основном — стандартизированное представление системных ресурсов, которыми может управлять набор демонов).

Units, в некотором смысле можно назвать службами или заданиями, однако юнит имеет гораздо более широкое определение, поскольку он может использоваться для абстрактных служб, сетевых ресурсов, устройств, монтирования файловой системы и изолированных пулов ресурсов.

Некоторые функции, которые легко реализовать:

- **Активация на основе сокетов:** Сокеты, связанные с сервисом, лучше всего вырываются из самого демона, чтобы обрабатываться отдельно. Это дает ряд преимуществ, - например, задержка запуска службы до тех пор, пока соответствующий сокет не будет доступен. Это также позволяет системе создавать все сокеты до начала запуска процесса, что позволяет параллельно загружать связанные службы.
- **Активация на основе шины:** Unit-ы также могут быть активированы на интерфейсе шины, предоставляемом D-Bus-ом. Устройство может быть запущено когда соответствующая шина доступна.
- **Активация на основе пути:** Unit может быть запущен на основе активности или наличия определенных путей к файловой системе. Это использует inotify.
- **Активация на основе устройства:** Unit-ы также могут быть запущены при первой доступности (подключении) связанного оборудования за счет использования событий udev.
- **Неявное сопоставление зависимостей:** Большая часть структуры зависимостей для юнитов может быть построена самой системой. Вы все равно можете добавить информацию о зависимостях, но большая часть тяжелой работы будет решена за вас.
- **Экземпляры и шаблоны:** Файлы блока шаблонов могут использоваться для создания нескольких экземпляров одного и того же общего устройства. Это позволяет создавать небольшие вариации или единичные unit-ы, которые обеспечивают одну и ту же общую функцию.
- **Простое упрощение безопасности:** Юниты могут реализовать некоторые довольно хорошие функции безопасности, добавив простые директивы. Например, вы можете указать какой доступ можно использовать ( чтение, запись) при работе с файловой системой, ограничить возможности ядра, установить приватный /tmp фолдер и сетевой доступ.
- **Drop-ins и snippets:** Units можно легко расширить, предоставив фрагменты, которые будут отменять части файла системы. Это позволяет легко переключаться между vanilla и индивидуальными реализациями.

Есть много других преимуществ, которые имеют системные юниты над рабочими элементами других систем, но это должно дать вам представление о мощности, которую можно использовать с помощью собственных конфигурационных директив.

### Как я могу найти/ Где находятся Systemd Unit в Unix/Linux?

Файлы, определяющие, как systemd будет обрабатывать unit, можно найти в разных местах, каждое из которых имеет разные приоритеты и смысл.

Все копии системных файлов системы (я имею ввиду всех юнитов) можно найти в /lib/systemd/system каталоге. Файлы, хранящиеся здесь, могут быть запущены и остановлены по требованию, но во время сеанса. Это будет общий файл ванильной единицы, который часто записывается сторонними разработчиками проекта, которые должны работать в любой системе, которая развертывает systemd в своей стандартной реализации. Вы не должны редактировать файлы в этом каталоге. Вместо этого вы должны переопределить файл (если необходимо) используя другое расположение файла, которое заменит файл в этом месте.

Если вы хотите изменить работу некоторого устройства, то для этого нужно перейти в /etc/systemd/system директорию, потому что — файлы, найденные в этом каталоге, имеют приоритет над любыми другими местами в файловой системе.

Если вы хотите переопределить только определенные директивы из системных unit файлов, вы можете фактически предоставить фрагменты unit файла в подкаталоге. Они будут добавлять или изменять директивами копии системы, позволяя указать только параметры, которые вы хотите изменить.

Правильный способ сделать это — создать каталог с именем после файла с добавлением «.d» в конце. Таким образом, для единицы под названием `my_test.service` нужно создать подкаталог `my_test.service.d`. В этом каталоге файл, заканчивающийся на `.conf`, может использоваться для переопределения или расширения атрибутов `unit` файла.

Также есть место для определения `run-time unit-ов` в `/run/systemd/system`. Файлы, найденные в этом каталоге, имеют приоритет между `/etc/systemd/system` и `/lib/systemd/system`.

Файлы в этом месте дают меньший приоритет, чем прежнее местоположение, но больший приоритет, чем последнее. Сам процесс `systemd` использует это местоположение для динамически создаваемых `unit` файлов, созданных во время выполнения. Этот каталог можно использовать для изменения поведения устройства системы в течение всего сеанса. Все изменения, внесенные в этот каталог, будут потеряны при перезагрузке сервера.

## Типы `systemd Unit` файлов

Самый простой способ определить тип устройства — это посмотреть на его суффикс, который добавляется к концу имени ресурса (юнита). В следующем списке описаны типы `unit-ов`, доступных для `systemd`:

- **.service**: Данный юнит описывает, как управлять службой или приложением на сервере. Он в себе будет включать запуск или остановку службы, при каких обстоятельствах она должна автоматически запускаться, а также информацию о зависимости для соответствующего программного обеспечения.
- **.socket**: Файл сокет-устройства описывает сетевой, IPC-сокет или FIFO буфер, который `systemd` использует для активации сокета. Они всегда имеют связанный `.service` файл, который будет запущен при сокет активности, которая определяет этот блок.
- **.device**: Юнит, который описывает устройство, которое было указано как необходимое для управления `systemd` с помощью `udev` или файловой системы `sysfs`. Не все устройства имеют файлы `.device`. Некоторые скрипты, которым может потребоваться устройства `.device`, предназначены для монтирования и доступа к устройствам.
- **.mount**: Этот блок определяет точку монтирования в системе, которой управляет `systemd`. Они называются по пути монтирования, при этом «/» изменены на тире. Записи внутри `/etc/fstab` могут иметь блоки, созданные автоматически.
- **.automount**: Модуль `.automount` настраивает точку монтирования, которая будет автоматически установлена и должен быть определен после точки монтирования, на которую они ссылаются, и должны иметь соответствующий модуль `.mount` для определения особенностей монтирования.
- **.swap**: Данный юнит, описывает SWAP (пространство подкачки) в системе. Название этих блоков должно отражать путь к устройству или файлу.
- **.target**: Данный юнит используется для обеспечения точек синхронизации для других устройств при загрузке или изменения состояний. Они также могут использоваться для перевода системы в новое состояние.
- **.path**: Данный юнит, определяет путь, который может использоваться для активации на основе пути. По умолчанию, юнит будет запущен с тем же базовым именем. Это использует `inotify` для отслеживания изменения путей.
- **.timer**: Этот юнит, определяет таймер, который будет управляться `systemd` (аналог `cron` джоб) для задержки или запланированной активации. При достижении таймера будет запускаться соответствующий блок.
- **.snapshot**: Это юнит, создается автоматически командой `snapshot systemctl`. Он позволяет вам восстановить текущее состояние системы после внесения изменений. Снимки не сохраняются во время сеансов и используются для отката временных состояний.

- Как вы можете видеть, существует множество различных юнитов с которыми взаимодействует система. Многие из них работают вместе, чтобы добавить функциональности. В основном, используется `.service` из-за их полезности.

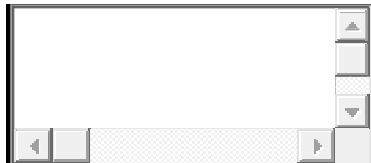
Внутренняя структура файлов организована с помощью разделов. Разделы обозначаются двумя квадратными скобками «[» и «]» с именем раздела, заключенного внутри. Каждый раздел продолжается до начала следующего раздела или до конца файла.

Названия разделов хорошо определены и учитывают регистр. Таким образом, раздел [Unit] не будет интерпретироваться правильно, если он записан как [UNIT]. Если вам нужно добавить нестандартные разделы для анализа (отличными от systemd), вы можете добавить X-префикс к имени раздела.

В этих разделах поведение устройства и метаданные определяются с помощью простых директив с использованием формата ключа-значения с назначением, обозначенным знаком равенства, например:

В случае переопределения файла (например, содержащегося в каталоге `unit.type.d`) директивы могут быть сброшены путем назначения пустой строкой. Например, копия единичного файла системы может содержать директиву, заданную таким образом:

1 lines

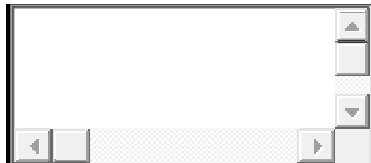


Directive1=default\_value

Значение default\_value можно исключить в файле переопределения, указав Directive1 (без значения), например:

```
sh
```

1 lines



Directive1=

Сейчас рассмотрим каждый из юнитов по отдельности.

## [Unit]

Первый раздел, найденный в большинстве юнит-файлов, — это раздел [Unit]. Обычно его используют для определения метаданных устройства и настройки отношения устройства к другим устройствам.

Хотя порядок разделов не имеет значения для systemd при парсинге файла, этот раздел часто размещается сверху, потому что он предоставляет обзор устройства. Некоторые общие директивы, которые вы найдете в разделе [Unit]:

- **Description=:** Эта директива может использоваться для описания имени и основных функций устройства. Он возвращается различными инструментами systemd, поэтому полезно установить что-то короткое, конкретное и информативное.
- **Documentation=:** Эта директива предоставляет местоположения документации. Это могут быть внутренние доступные справочные страницы, либо доступные в интернете (URL-адреса). Команда «status system your\_service» выведет эту информацию.
- **Requires=:** В этой директиве перечислены все юниты, от которых зависит этот юнит (служба или устройство). Если текущий блок активирован, перечисленные здесь юниты также должны успешно активироваться, иначе он потерпит неудачу. Эти блоки запускаются параллельно с текущим устройством по умолчанию.

- **Wants=**: Эта директива похожа на Requires=, но менее строгая. Systemd будет пытаться запустить любые юниты, перечисленные здесь, когда это устройство активировано. Если эти устройства не обнаружены или не запущены, текущий блок будет продолжать функционировать. Это рекомендуемый способ настройки большинства зависимостей. Опять же, это подразумевает параллельную активацию, если не изменено другими директивами.
- **BindsTo=**: Эта директива аналогична Requires =, но также приводит к остановке текущего устройства, когда соответствующий узел завершается.
- **Before=**: Юниты, перечисленные в этой директиве, не будут запущены до тех пор, пока текущий блок не будет отмечен как запущенный, если они будут активированы одновременно.
- **After=**: Юниты, перечисленные в этой директиве, будут запущены до запуска текущего устройства (юнита).
- **Conflicts=**: Данный юнит можно использовать для отображения юнитов, которые нельзя запускать одновременно с текущим устройством. Запуск устройства с этой связью приведет к остановке других устройств.
- **Condition...=**: Существует ряд директив, начинающихся с условия, которые позволяют администратору протестировать определенные условия до запуска устройства. Это можно использовать для предоставления файла универсального элемента, который будет запускаться только в соответствующих системах. Если условие не выполнено, юнит — пропускается.
- **Assert...=**: Подобно директиве что выше (Condition...=), но установленные директивы проверяют различные аспекты рабочей среды, чтобы решить, следует ли активировать устройство. Однако, в отличие от директив Condition...=, отрицательный результат вызывает сбой в этой директиве.

Используя эти директивы и несколько других, можно установить общую информацию об устройстве и его взаимосвязь в операционной системой.

## [Install]

В самом конце файла, расположен — [Install] секция. Этот раздел является дополнительным и используется для определения поведения или юнита, если он включен или отключен. Включение устройства означает, что он автоматически запускается при загрузке. По сути, это достигается путем фиксации рассматриваемого устройства на другом блоке, который находится где-то в строке единиц, которые нужно запустить при загрузке.

Из-за этого, только юниты, которые могут быть включены, будут иметь этот раздел. Директивы внутри, говорят что должно произойти, когда устройство включено:

- **WantedBy=**: Данная директива, является наиболее распространенным способом определения того, как устройство должно быть включено. Эта директива позволяет вам указать зависимость (аналогично директиве Wants = в разделе [Unit]). Разница в том, что эта директива включена в вспомогательную единицу, позволяющую первичному блоку оставаться относительно чистым. Когда устройство с этой директивой определено, то будет создана в системе папка в /etc/systemd/, названная в честь указанного устройства, но с добавлением «.wants» в самом конце. В этом случае будет создана символическая ссылка на текущий блок, создающий зависимость. Например, если текущий блок имеет WantedBy = multi-user.target, каталог с именем multi-user.target.wants будет создан в /etc/systemd/system (если он еще не доступен) и символическая ссылка на текущий блок будут размещены внутри. Отключение этого устройства удаляет связь и удаляет зависимость.
- **RequiredBy=**: Эта директива очень похожа на директиву WantedBy =, но вместо этого указывает требуемую зависимость, которая приведет к сбою активации, если не будет выполнена. Когда включено, юнит с этой директивой создаст каталог, заканчивающийся на .requires.
- **Alias=**: Эта директива позволяет включить устройство под другим именем. Среди других применений это позволяет доступным нескольким провайдерам функции, чтобы соответствующие подразделения могли искать любого провайдера с общим псевдонимом.
- **Also=**: Эта директива позволяет включать или отключать блоки в качестве набора. Здесь можно указать поддерживающие устройства, которые всегда должны быть доступны, когда это устройство активно. Они будут управляться как группа для задач установки.

- **DefaultInstance=:** Для template units (более поздних), которые могут создавать экземпляры блоков с непредсказуемыми именами, это может использоваться как резервное значение для имени, если соответствующее имя не предоставляется.

## [Service]

Раздел [Service] используется для предоставления конфигурации, которая применима только для служб. Одной из основных вещей, которые должны быть указаны в разделе [Service], является Type= служба. Это классифицирует услуги по их процессу и демонизирующему поведению. Это важно, потому что он сообщает systemd, как правильно управлять службой и узнать его состояние.

Директива Type= может быть одной из следующих:

- **simple:** Основной процесс службы указан в стартовой строке. Это значение по умолчанию, если директивы Type= и Busname= не установлены, но установлен ExecStart=. Любое сообщение должно обрабатываться вне устройства через второй блок соответствующего типа (например, через блок .socket, если эта служба должна обмениваться данными с помощью сокетов).
- **forking:** Этот тип службы используется, когда служба форкает дочерние процессы при мгновенном покидании родительского процесса. Это сообщает systemd, что процесс все еще работает, даже несмотря на то, что родитель завершил сеанс. Хорошо подходит, например для запуска php-fpm, nginx, tomcat.
- **oneshot:** Этот тип указывает, что процесс будет недолговечным и система должна ждать завершения процесса, прежде чем продолжить работу с другими устройствами. Значение по умолчанию для Type= и ExecStart= не установлены. Он используется для одnorазовых задач.
- **dbus:** Это означает, что устройство будет использовать имя от D-Bus шины. Когда это произойдет, systemd продолжит обработку следующего блока.
- **notify:** Это указывает на то, что служба отправит уведомление, когда закончит запуск. Процесс systemd будет ждать, пока это произойдет, прежде чем переходить к другим устройствам.
- **idle:** Это означает, что служба не будет запущена до тех пор, пока не будут отправлены все задания.

При использовании определенных типов служб, могут потребоваться некоторые дополнительные директивы. Например:

- **RemainAfterExit=:** Эта директива обычно используется с типом oneshot. Это означает, что сервис следует считать активным даже после выхода из процесса.
- **PIDFile=:** Если тип службы отмечен как «forking», эта директива используется для установки пути файла, который должен содержать идентификационный номер процесса основного «ребенка», который должен контролироваться.
- **BusName=:** Эта директива должна быть установлена на имя шины D-Bus, которое служба попытается получить при использовании типа службы «dbus».
- **NotifyAccess=:** Эта директива указывает на доступ к сокету, который должен использоваться для прослушивания уведомлений при выборе «notify». Она может быть «none», «main» или «all». По умолчанию «none» игнорирует все сообщения о состоянии. «main» вариант будет прослушивать сообщения из основного процесса, а опция «all» приведет к обработке всех членов контрольной группы службы.

До сих пор мы обсуждали некоторую предварительную информацию, но фактически не говорили, как управлять нашими услугами. Для этого имеются следующие директивы:

- **ExecStart=:** Нужно указать полный путь и аргументы команды, которая должна быть выполнена для запуска процесса. Это может быть указано только один раз (кроме сервисов «oneshot»). Если для пути к команде предшествует символ «-» тире, будут приняты ненулевые статусы выхода без маркировки активации устройства как сбоя.

- **ExecStartPre=**: Нужно указать полный путь и аргументы команды, которые должны быть выполнены до запуска основного процесса. Это можно использовать несколько раз.
- **ExecStartPost=**: Это имеет те же самые качества, что и ExecStartPre = за исключением того, что данная директива указывает команды, которые будут запускаться после запуска основного процесса.
- **ExecReload=**: Эта необязательная директива указывает команду, необходимую для перезагрузки конфигурации службы, если она доступна.
- **ExecStop=**: Это указывает на команду, необходимую для остановки службы. Если это не указано, процесс будет немедленно уничтожен, когда служба будет остановлена.
- **ExecStopPost=**: Это можно использовать для указания команд для выполнения после остановки команды.
- **RestartSec=**: Если автоматический перезапуск службы включен, это указывает время ожидания перед попыткой перезапуска службы.
- **Restart=**: Это указывает на обстоятельства, при которых systemd будет пытаться автоматически перезапустить службу. Она может быть установлена как значения «always», «on-success», «on-failure», «on-abnormal», «on-abort» или «on-watchdog». Это приведет к перезапуску в соответствии с тем, как служба была остановлена.
- **TimeoutSec=**: Это устанавливает время, в течение которого система будет ждать остановки службы, прежде чем пометить ее как неудачную или убитую принудительно. Вы можете установить отдельные таймауты с помощью TimeoutStartSec = и TimeoutStopSec =.

## [Socket]

Socket очень часто встречаются в конфигурациях systemd, потому что многие службы реализуют активацию на основе сокетов, чтобы обеспечить лучшую распараллеливание и гибкость. Каждый блок socket-а должен иметь соответствующий сервисный модуль, который будет активирован, когда сокет получает активность.

Разрушая управление сокета вне самой службы, сокеты могут быть инициализированы раньше, и связанные службы могут часто запускаться параллельно. По умолчанию имя сокета будет пытаться запустить службу с тем же именем после получения соединения. Когда служба инициализируется, сокет будет передан ему, что позволит ему начать обработку любых буферизованных запросов.

Чтобы указать фактический сокет, эти директивы являются общими:

- **ListenStream=**: Это определяет адрес для сокет stream, который поддерживает последовательную, надежную связь. Службы, использующие TCP, должны использовать этот тип сокета.
- **ListenDatagram=**: Это определяет адрес для сокет datagram, который поддерживает быстрые, ненадежные пакеты связи. Службы, использующие UDP, должны устанавливать этот тип сокета.
- **ListenSequentialPacket=**: Это определяет адрес для последовательной, надежной связи с максимальными дейтаграммами, которые сохраняют границы сообщений. Это чаще всего встречается для Unix сокетов.
- **ListenFIFO**: Наряду с другими типами прослушивания вы также можете указать буфер FIFO вместо сокета.

Существует больше типов директив для установки соединений, но те которые указаны выше, являются наиболее распространенными.

Другие характеристики сокетов можно контролировать с помощью дополнительных директив:



- **Accept=:** Это определяет, будет ли запущен дополнительный экземпляр службы для каждого соединения. Если установлено значение false (по умолчанию), один экземпляр будет обрабатывать все соединения.
- **SocketUser=:** С помощью Unix-сокета задается его владелец. Если не указать, то будет пользователь root.
- **SocketGroup=:** С помощью Unix-сокета задается владелец группы. Это будет root группа, если не указано ни то, ни другое. Если установлен только SocketUser =, systemd попытается найти подходящую группу.
- **SocketMode=:** Для сокетов Unix или буферов FIFO это устанавливает разрешения для созданного объекта.
- **Service=:** Если имя службы не совпадает с именем **.socket**, эта служба может быть указана с помощью этой директивы.

## [Mount]

Модули монтирования позволяют управлять точкой монтирования изнутри systemd. Точки монтирования называются в соответствии с управляемым им каталогом с применением алгоритма перевода.

Например, ведущая косая черта (слеш или «/») удаляется, все остальные косые черты переводится в тире «-», и все тире и непечатаемые символы заменяются escape-кодами стиля C. Результат этого перевода используется как имя узла монтирования. Unit-ы монтирования будут иметь неявную зависимость от других монтировок над ней в иерархии.

Mount юниты часто переводятся непосредственно из файла /etc/fstab во время процесса загрузки. Для автоматически созданных определений единиц и те, которые вы хотите определить в единичном файле:

- **What=:** Абсолютный путь к ресурсу, который необходимо смонтировать.
- **Where=:** Абсолютный путь точки монтирования, в которой должен быть установлен ресурс. Это должно быть таким же, как имя файла устройства, за исключением использования стандартной записи файловой системы.
- **Type=:** Тип файловой системы для монтирования.
- **Options=:** Любые параметры монтирования, которые необходимо применить. Это список, разделенный запятыми.
- **SloppyOptions=:** Логическое значение (0 или 1), которое определяет произойдет ли сбой монтирования, если есть параметр непризнанного монтирования.
- **DirectoryMode=:** Если родительские каталоги необходимо создать для точки монтирования, это определяет режим разрешений этих папок.
- **TimeoutSec=:** Настраивает время, в течение которого система будет ждать, пока операция монтирования не будет отмечена как сбой.

## [Automount]

Этот модуль позволяет автоматически устанавливать подключенный модуль .mount при загрузке. Как и в случае с модулем .mount, эти юниты должны быть названы в честь пути переведенной точки монтирования.

[Automount] Раздел довольно прост, разрешены только следующие два варианта:

- **Where=:** Абсолютный путь automount в файловой системе. Это будет соответствовать имени файла, за исключением того, что вместо перевода используется условное обозначение пути.

- **DirectoryMode=**: Если необходимо создать automount или любые родительские каталоги, это определит настройки разрешений для этих компонентов пути.

## [Swap]

Swap модули используются для настройки подкачки (свопинга) в системе. Юниты должны быть названы по названию файла или устройства подкачки, используя тот же перенос файловой системы, о котором говорилось выше.

Как и mount параметры, блоки swap могут быть автоматически созданы из /etc/fstab или могут быть сконфигурированы через выделенный unit файл.

Раздел [Swap] может содержать следующие директивы для конфигурации:

- **What=**: Абсолютный путь к местоположению подкачки (будь то файл или устройство).
- **Priority=**: Данная опция принимает целое число, которое задает приоритет для настройки подкачки.
- **Options=**: Любые параметры, которые обычно задаются в файле /etc/fstab, могут быть установлены с помощью этой директивы. Используется список, разделенный запятыми.
- **TimeoutSec=**: Данная опция задает времени, в течение которого, система ожидает активацию своп-а, прежде чем отмечать операцию как зафейленную.

## [Path]

Блок path, определяет путь файловой системы, который systemd может отслеживать изменения. Должен существовать другой блок, который будет активирован, когда определенная активность будет обнаружена в местоположении пути. Активность пути определяется тем, что он не влияет на события.

Раздел [Path] может содержать следующие директивы:

- **PathExists=**: Эта директива используется для проверки того, существует ли этот путь. Если путь существует, активируется соответствующий блок.
- **PathExistsGlob=**: Данная опция такая как и выше, но поддерживает глобальные выражения для определения существования пути.
- **PathChanged=**: Данную опцию используют для отслеживания изменений местоположение пути. Связанный блок активируется, если обнаружено изменение (проверяется когда файл закрыт).
- **PathModified=**: Данная опция такая как и выше, но активируется при записи файлов, а также когда файл закрыт.
- **DirectoryNotEmpty=**: Эта директива позволяет systemd активировать связанный блок, когда каталог больше не пустой.
- **Unit=**: Это указывает, что устройство активируется, когда условия пути, указанные выше, выполняются. Если это будет опущено, systemd будет искать файл .service, который имеет то же имя базового юнита, что и этот блок.
- **MakeDirectory=**: Данная директива определяет, будет ли systemd создавать структуру каталогов перед просмотром.
- **DirectoryMode=**: Если вышеуказанная директива включена, то данная опция установит режим разрешения любых компонентов пути, которые должны быть созданы.

## [Timer]

Timer юнит, используются для планирования задач для работы в определенное время или после определенной задержки. Этот тип устройства заменяет или дополняет некоторые функции cron-а и демонов. Должен быть предоставлен соответствующий блок, который будет активирован, когда таймер будет достигнут.

Раздел [Timer] может содержать некоторые из следующих директив:

- **OnActiveSec=**: Эта директива позволяет активировать соответствующий блок относительно активации модуля .timer.
- **OnBootSec=**: Эта директива задает время, когда будет активироваться соответствующее устройство после загрузки системы.
- **OnStartupSec=**: Эта директива аналогична указанному выше таймеру, но задает когда будет активироваться systemd процесс после загрузки системы.
- **OnUnitActiveSec=**: Это устанавливает таймер в зависимости от того, когда последний активировался.
- **OnUnitInactiveSec=**: Это устанавливает таймер в отношении того когда unit был неактивный.
- **OnCalendar=**: Это позволяет активировать соответствующий блок путем определения абсолютного (вместо относительно).
- **AccuracySec=**: Данный юнит используется для установки уровня точности с таймером, который должен быть приклеен. По умолчанию, связанный с ним блок будет активирован в течение одной минуты.
- **Unit=**: Эта директива используется для указания того, что необходимо активировать когда таймер истечет. Если не установлены, Systemd будет искать блок с именем .service.
- **Persistent=**: Если это установлено, Systemd запустит триггер соответствующего блока.
- **WakeSystem=**: Установка этой директивы позволяет «будить систему» из режима ожидания, если таймер будет достигнут.

## [Slice]

Раздел [Slice] на самом деле, не имеет .slice специфической конфигурации. Вместо этого он может содержать некоторые директивы управления ресурсами, которые перечислялись выше.

Хватит теории, перейдем к практике.

## Пишем systemd Unit файл

Первое что стоит сделать — так это проверить инициализацию:

```
sh
```

1 lines



```
# ps -s1 | awk '{print $4}' | grep -Ev "CMD"
```

PS: Вот небольшая статья:

### [Система инициализации в Unix/Linux](#)

Файлы шаблонов блоков могут быть определены, потому что они содержат символ @ после имени базового блока и перед суффиксом блокового типа. Имя файла блока с шаблоном может выглядеть следующим образом:

```
sh
```

1 lines



```
my_test@.service
```

Из созданного блока ( что выше) можно создать экземпляр др блока, который выглядит следующим образом:

```
sh
```

1 lines



```
my_test@instance1.service
```

Мощность файлов шаблонных модулей в основном проявляется благодаря возможности динамически подставлять соответствующую информацию в определение устройства в соответствии со средой (ENV). Это делается путем установки директив в файл шаблона как обычно, но заменяя определенные значения или части значений спецификаторами переменных.

Ниже приведены некоторые из наиболее распространенных спецификаторов, которые будут заменены, когда юнит-экземпляра интерпретируется с соответствующей информацией:

- **%n**: В любом месте, где это используется, будет добавлено полное имя элемента.
- **%N**: Это то же самое, что и выше, но любое экранирование, такое как те, что присутствуют в шаблонах пути файла, будет отменено.
- **%p**: Это указывает префикс имени юнита. Это часть названия юнита, которая предшествует символу **@**.
- **%P**: Это то же самое что и выше, но с любым отступлением.
- **%i**: Это ссылается на имя экземпляра, которое является идентификатором, следующим за **@** в экземпляре. Это один из наиболее часто используемых спецификаторов, поскольку он динамичный. Использование этого идентификатора поощряет использование значимых идентификаторов конфигурации. Например, порт, в котором будет выполняться служба, может использоваться как идентификатор экземпляра, и шаблон может использовать этот спецификатор для настройки спецификации порта.
- **%I**: Этот спецификатор такой же, как и выше, но с любым отступлением.
- **%f**: Это будет заменено на имя неэкранированного юнита или имя префикса, добавленное к «/».
- **%c**: Это будет указывать на управляющую группу устройства со стандартной родительской иерархией **/sys/fs/cgroup/ssytemd**.
- **%u**: Имя пользователя, настроенного для запуска юнита.
- **%U**: То же, что и выше, **UID** вместо имени.
- **%H**: Имя хоста системы, на котором выполняется юнит.
- **%%**: Это используется для вставки буквенного знака процента.

Используя приведенные выше идентификаторы в шаблоне файла, Systemd заполнит правильные значения при интерпретации шаблона для создания юнит-экземпляра.

## Примеры systemd Unit файлов

Рассмотрим следующий пример, — это написания systemd скрипта для запуска tomcat-a. Для этого, открываем файл:

```
sh
```

1 lines



```
# vim /etc/systemd/system/tomcat9.service
```

И записываем в него следующий код:

```
sh
```

22 lines



```
[Unit]
```

```
Description=Tomcat9
```

```
After=syslog.target network.target
```

```
[Service]
```

```
Type=forking
```

```
User=tomcat
```

```
Group=tomcat
```

```
Environment=CATALINA_PID=/usr/local/tomcat9/tomcat9.pid
```

```
Environment=TOMCAT_JAVA_HOME=/usr/bin/java
```

```
Environment=CATALINA_HOME=/usr/local/tomcat9
```

```
Environment=CATALINA_BASE=/usr/local/tomcat9
```

```
Environment=CATALINA_OPTS=
```

```
#Environment=CATALINA_OPTS=-Xms512M -Xmx1024M -server -XX:+UseParallelGC
```

```
Environment="JAVA_OPTS=-Dfile.encoding=UTF-8 -Dnet.sf.ehcache.skipUpdateCheck=true -XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled -XX:+UseParNewGC -XX:MaxPermSize=128m -Xms512m -Xmx512m"
```

```
ExecStart=/usr/local/tomcat9/bin/startup.sh
```

```
ExecStop=/bin/kill -15 $MAINPID
```

```
[Install]
```

```
WantedBy=multi-user.target
```

Перезагрузим службу:

sh

1 lines



```
# systemctl daemon-reload
```

Добавим томкат в автозагрузку ОС:

sh

1 lines



```
# systemctl enable tomcat9
```

Перезапускаем томкат:

sh

1 lines




```
# systemctl restart tomcat9
```

Чтобы проверить статус, выполняем:

sh

13 lines



```
# systemctl status tomcat9
● tomcat9.service - Tomcat9
   Loaded: loaded (/etc/systemd/system/tomcat9.service; enabled)
   Active: active (running) since Tue 2017-05-09 22:04:58 EEST; 6s ago
     Process: 28528 ExecStop=/bin/kill -15 $MAINPID (code=exited, status=0/SUCCESS)
     Process: 28531 ExecStart=/usr/local/tomcat9/bin/startup.sh (code=exited, status=0/SUCCESS)
    Main PID: 28541 (java)
      CGroup: /system.slice/tomcat9.service
              └─28541 /usr/bin/java -Djava.util.logging.config.file=/usr/local/tomcat9/conf/logging.properties -Djava.ut...
May 09 22:04:58 debian systemd[1]: Starting Tomcat9...
May 09 22:04:58 debian startup.sh[28531]: Tomcat started.
May 09 22:04:58 debian systemd[1]: Started Tomcat9.
```

Как видим, все четко работает. На этом, у меня все! Больше примеров будет дальше. Я буду добавлять их по мере необходимости. Но в основном, я использую шаблон что выше ( только немного его видоизменяю).

Статья «Пишем systemd Unit файл» завершена.