Семинар #9: systemd

Команды `systemctl`

Для некоторых команд необходимы права суперпользователя, то есть нужно использовать `sudo`.

Команда	Назначение
`systemctl start <unit>`	Запускает юнит.
`systemctl stop <unit>`	Останавливает юнит.
`systemctl restart <unit>`	Перезапускает юнит.
`systemctl reload <unit>`	Перечитать конфиг юнита без перезапуска.
`systemctl enable <unit>`	Включить автозапуск юнита при загрузке.
`systemctl disable <unit>`	Отключить автозапуск юнита при загрузке.
`systemctl status <unit>`	Отображает состояние юнита, логи, PID и другую информацию.
`systemctl cat <unit>`	Печатает путь до юнит-файла и сам юнит-файл.
`systemctl edit <unit>`	Редактировать директивы юнита.
`systemctl show <unit>`	Показывает свойства юнита.
`systemctl is-active <unit>`	Проверяет, активен ли юнит.
`systemctl is-failed <unit>`	Проверяет, находится ли юнит в состоянии "failed".
`systemctl is-enabled <unit>`	Проверяет, включён ли юнит для автозапуска.
`systemctl`	Показывает активные юниты.
`systemctl --all`	Показывает все юниты, даже неактивные.
`systemctl list-units --type=service`	Показывает активные сервисы.
`systemctl daemon-reload`	Перечитать все юнит-файлы. Нужно выполнить после редактирования юнит-файлов.

Основные типы юнитов

Тип юнита	Назначение
`service`	Запуск и управление службами/демонами.
`target`	Группа юнитов.
`timer`	Юнит, который активирует сервисы по расписанию (альтернатива cron).
`path`	Следит за изменениями файлов/директорий и активирует сервисы.
`socket`	Управление сокет-активацией.

Ещё типы юнитов: `device`, `mount`, `automount`, `slice`, `scope`, `swap`, `busname`, `snapshot` и другие.

Алгоритм поиска юнит файлов

Имена юнит файлов имеют вид `<имя юнита>.<тип юнита>`, например сервис под названием `nginx` имеет юнит-файл под названием `nginx.service`. systemd ищет юнит файлы по следующему алгоритму:

1. Сначала ищет в директории `/etc/systemd/system`.
2. Если не нашёл, то ищет в директории `/run/systemd/system`.
3. Если не нашёл, то ищет в директории `/usr/lib/systemd/system`.

Плюс, существуют ещё некоторые файлы, в которых можно переопределить директивы юнита или скрыть юнит.

Секции и директивы в юнит файлах

Секция	Назначение
`[Unit]`	Используется во всех типах юнитов. Задает общую информацию и зависимости.
`[Service]`	Используется только в `service`-юнитах.
`[Install]`	Описывает, как юнит "встраивается" в систему при `systemctl enable`.
`[Timer]`	Используется только в `timer`-юнитах.
`[Тип_юнита]`	Используется только в `Тип_юнита`-юнитах.

Директивы секции `[Unit]`

- `Description`
  Короткое описание юнита.

- `Documentation`
  Ссылка на документацию.

- `After`
  Устанавливает порядок запуска юнита. В этой директиве указывается один или несколько других юнитов (через пробел). Если данный юнит запускается вместе с теми, что перечислены в `After`, то он будет запущен после всех этих юнитов. Но если эти юниты не запускаются вместе с данным, то они не будут дополнительно запущены.

- `Before`
  Устанавливает порядок запуска юнита. Как `After`, но только данный юнит будет запущен до тех, которые указаны в `Before`.

- `Wants`
  Мягкие зависимости юнита. В этой директиве указывается один или несколько других юнитов (через пробел). Если при запуске данного юнита, юниты, указанные в `Wants` не запускаются, то `systemd` попытается их запустить. Если запустить юнит/юниты в `Wants` не получится, то данный юнит всё-равно может быть успешно запущен.

- `Requires`
  Жесткие зависимости юнита. Как `Wants`, но если хотя бы один из зависимых юнитов запустить не получилось, то данный юнит тоже не запустится и будет в состоянии `failed`.

- `Conflicts`
  Данный юнит не может работать одновременно с теми, что указаны в `Conflicts`. При старте данного юнита, юниты указанные в `Conflicts` останавливаются.

- `Condition*`
  Директивы, начинающиеся с `Condition` задают некоторое условие. Если хотя бы одно условие не выполняется, то systemd останавливает дальнейшую активацию данного юнита и помечает его как `skipped / inactive`. Примеры таких директив:

  - `ConditionPathExists` – проверяет, существует ли файл по указанному пути.
  - `ConditionFileNotEmpty` – файл существует, является обычным файлом и этот файл не пуст.
  - `ConditionFileIsExecutable` – файл существует, является обычным файлом, и этот файл имеет право на исполнение.
  - `ConditionACPower` – `true` если питание от сети и `false` иначе.
  - `ConditionUser`/`ConditionGroup` – проверяет наличие указанного пользователя или группы в системе.
  - `ConditionCapability` – проверяет, есть ли у systemd определённая Linux-capability.

- `Assert*`
  Директивы, начинающиеся с `Assert` задают некоторое условие, аналогичные условиям, задаваемым с помощью `Condition`. Но, если хотя бы одно из таких условий не выполняется, то юнит не запускается и считается `failed`.

- `OnFailure`
  В этой директиве указываются юниты, которые запускаются при переходе данного юнита в состояние `failed`.

Директивы секции `[Service]`

- `Type`
  Тип сервиса, определяет модель запуска сервиса. Существуют следующие типы:
  - `simple`
    Запускается напрямую через команду в `ExecStart`. Сервис считается запущенным сразу после старта процесса. Это тип сервиса по умолчанию.
  - `forking`
    Сначала запускается процесс через команду в `ExecStart`. Предполагается, что запущенный процесс создаст дочерний процесс через `fork`, после чего сам завершится. systemd будет отслеживать дочерний процесс и считать его сервисом. Сервис начинает считаться запущенным, когда systemd узнаёт его PID. Чтобы узнать PID сервиса (то есть PID дочернего процесса) используется `PIDFile` или другие способы.
  - `oneshot`
    Запускает команду в `ExecStart` и ждёт её выполнения. Сервис считается "активным" только после выполнения команды в `ExecStart`. Используется для одноразовых задач, которые выполняют задачу один раз и завершают работу.
  - `notify`
    Запускается напрямую через команду в `ExecStart`. Сервис считается запущенным когда он сам сообщит об этом systemd с помощью функции `sd_notify`. Используется для сервисов с долгой инициализацией.
  - `dbus`
  - `idle`
- `ExecStart`
  Основная команда запуска сервиса. Выполняется при запуске сервиса, то есть при

  ```
  $ sudo systemctl start <service>
  ```

  Но что конкретно будет происходить при старте сервиса определяется типом сервиса.
- `ExecStop`
  Команда, выполняемая при остановке сервиса (`systemctl stop`). Если не указано, то systemd просто посылает сигнал SIGTERM основному процессу.
- `ExecReload`
  Команда, выполняемая для перезагрузки конфигурации сервиса. Вызывается при использовании (`systemctl reload`). Если не указано, то команда `systemctl reload` будет недоступна для данного сервиса.
- `ExecStartPre`
  Команды, выполняемые перед запуском основного процесса (перед `ExecStart`). Используется для подготовки к запуску сервиса (создание директорий, монтирование, проверка конфигураций). Если команда завершается с ошибкой, сервис не запускается.
- `ExecStartPost`
  Команды, выполняемые после запуска основного процесса (после `ExecStart`). Ошибка в `ExecStartPost` не останавливает основной процесс, но сервис может считаться неуспешным.
- `ExecStopPost`
  Команды, выполняемые после остановки сервиса. Используется для очистки ресурсов, удаления временных файлов и т. д.
- `Restart`
  Определяет, будет ли перезапускаться сервис, если он завершился (успешно или с ошибкой). Может принимать следующие значения:

  - `no`
    Сервис не перезапускается автоматически.
  - `always`
    Сервис всегда перезапускается.
  - `on-success`
    Сервис перезапускается, только если он завершился успешно.
  - `on-failure`
    Сервис перезапускается, только если он завершился с ошибкой. Под ошибкой тут понимается:
    * Завершение сервиса с ненулевым кодом возврата.
    * Если процесс был убит сигналом.
    * Если процесс не успел стартовать или остановиться в пределах `TimeoutStartSec` / `TimeoutStopSec`.
    * Если задан `WatchdogSec` и сервис не отправляет heartbeat через `sd_notify`.
  - `on-abnormal`
    Сервис перезапускается, только если он завершился ненормально, а именно:
    * Был убит сигналом.
    * Завершился по таймауту (`TimeoutStartSec`, `TimeoutStopSec`).
    * Процесс завершился аварийно (обычно из-за ошибки вроде сегментационной ошибки — segmentation fault) и ядро системы создало дамп памяти.
  - `on-abort`
    Сервис перезапускается, только если он завершился по сигналу `SIGABRT`.
  - `on-watchdog`
    Сервис перезапускается, только если задан `WatchdogSec` и сервис не отправляет heartbeat через `sd_notify`.

- `RestartSec`
  Задержка перед перезапуском сервиса при использовании `Restart`.

  ```
  RestartSec=5s
  ```

  Можно использовать суффиксы: `ms`, `s`, `min`, `h`. По умолчанию используется задержка `100ms`. Для ресурсоёмких сервисов ставится задержка побольше (10–30 секунд), чтобы избежать перегрузки системы при постоянных падениях.

- `User`/`Group`
  Сервис запускается от имени указанного пользователя/группы.

- `Environment`, `EnvironmentFile`
  С помощью этих директив можно задать переменные среды запускаемого процесса.

- `WorkingDirectory`
  Задаёт рабочую директорию процесса.

- `CapabilityBoundingSet`
  Задаёт capabilities (способности) для процессов сервиса. Capabilities – это механизм, который разбивает полномочия суперпользователя (root) на отдельные, более мелкие права. По умолчанию systemd запускает сервис с полным набором способностей, если он работает от `root`. Задать способности сервиса можно, если задать директиву `CapabilityBoundingSet`. В ней прописываются разрешённые способности процесса. Способности могут быть следующими:

Capability	Назначение
CAP_SYS_ADMIN	Монтирование файловых систем, настройки namespaces, управления cgroups.
CAP_CHOWN	Изменение владельца файлов.
CAP_DAC_OVERRIDE	Игнорирование стандартных прав доступа к файлам.
CAP_SETUID	Позволяет менять UID процесса.
CAP_SETGID	Позволяет менять GID процесса.
CAP_NET_BIND_SERVICE	Позволяет процессу слушать порты <1024, например 80 (HTTP) или 443 (HTTPS).
CAP_NET_RAW	Позволяет создавать raw-сокеты.
CAP_MKNOD	Позволяет создавать специальные device файлы.

Все способности можно посмотреть в `https://man7.org/linux/man-pages/man7/capabilities.7.html`.

- `PIDFile`
  Эта директива нужна для сервисов типа `forking`. Указывает путь до файла, в который родительский процесс записывает PID дочернего процесса. Таким образом systemd и другие процессы в системе могут узнать PID демона.

- `StandardOutput` и `StandardError`
  Управляет тем, куда перенаправляется стандартный вывод (stdout) или стандартный поток ошибок (stderr) процесса.

  - `inherit`
    Наследует настройки вывода от родительского процесса (обычно от systemd).
  - `journal`
    Отправляет вывод в systemd-journal. Это поведение по умолчанию.
  - `console`
    Печатает вывод на системную консоль.
  - `null`
    Отбрасывает всё, что сервис печатает в stdout.

- `TimeoutStartSec`
  Задаёт максимальное время, которое systemd будет ждать успешного запуска сервиса. Если сервис не запустился за указанное время, то процессу будет послан сигнал SIGTERM (если не помогло, то SIGKILL) и сервис перейдёт в состояние `failed`. Значение по умолчанию можно задать в `/etc/systemd/system.conf`. Если в этом файле значения по умолчанию не заданы, то systemd берёт внутреннее значение по умолчанию, которое обычно равно 90 секунд.

- `TimeoutStopSec`
  Максимальное время, которое systemd будет ждать корректного завершения сервиса при остановке. Если процесс не завершился за заданное время, то systemd отправляет SIGKILL.

- `KillMode`
  Управляет тем, какие процессы будут завершены при остановке сервиса. Может принимать следующие значения:

  - `control-group`
    Значение по умолчанию. Завершаются все процессы в cgroup сервиса (главный и дочерние процессы). Сначала всем процессам посылается сигнал, заданный в `KillSignal`, но если за время `TimeoutStopSec` процессы не завершились, то посылается SIGKILL.
  - `process`
    Завершается только главный процесс (тот, что systemd считает основным). Дочерние процессы остаются жить. Сначала процессу посылается сигнал, заданный в `KillSignal`, но если за время `TimeoutStopSec` он не завершился, то посылается SIGKILL.
  - `mixed`
    Главному процессу посылается сигнал, заданный в `KillSignal`, а всем остальным в cgroup сразу SIGKILL. Используется, если дочерние процессы могут зависнуть, а главный должен завершиться корректно.

- `none`
  systemd не посылает никаких сигналов. Завершение процессов полностью остаётся на логике сервиса. systemd просто меняет статус юнита на `stopped` или `inactive`, но процессы продолжают жить пока сами не завершатся.

- `KillSignal`
  Определяет, какой сигнал будет отправлен процессу при его остановке. Значение по умолчанию SIGTERM.

Директивы секции `[Install]`

- `WantedBy`
  Говорит systemd, в какой target должен быть включён юнит при выполнении `systemctl enable`. Например, если написать:

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

  то данный юнит будет запускаться, когда активируется юнит `multi-user.target`. `multi-user.target` активируется после загрузки системы, когда система готова к многопользовательской работе в текстовом режиме. При загрузке системе в графическом режиме (`graphical.target`) сначала активируется `multi-user.target`, а затем запускаются графические службы.

- `RequiredBy`
  Работает как `WantedBy`, но связь жёсткая: юнит, который указывает `RequiredBy`, не запустится без этого юнита.

- `Also`
  Позволяет указать дополнительные юниты, которые должны быть `enable/disable` вместе с этим.

- `Alias`
  Создаёт дополнительные имена (alias) для юнита.

Директивы секции `[Timer]`

Таймеры systemd (*.timer) запускают соответствующие сервисы (*.service) по заданному расписанию. Имя таймера и сервиса должны совпадать.

- `Unit`
  Задаёт имя юнита, запускаемого этим таймером. По умолчанию таймер запускает юнит с тем же именем, что у таймера.

- `OnBootSec`
  Задаёт время, через который должен быть запущен связанный сервис после загрузки системы.

- `OnActiveSec`
  Задаёт время, через который должен быть запущен связанный сервис после активации таймера (после `systemctl start my.timer`).

- `OnUnitActiveSec`
  Задаёт время, через который должен быть запущен связанный сервис после последнего успешного запуска связанного сервиса через `systemctl start my.service`.

- `Persistent`
  Если `Persistent=yes`, то пропущенные срабатывания (например, когда компьютер был выключен) будут выполнены сразу после включения.

- `RandomizedDelaySec`
  Добавляет случайную задержку при запуске сервиса. Задаётся максимальное значение этой случайной задержки. Используется для того, чтобы избежать ситуации, когда множество таймеров или сервисов стартуют одновременно и создают нагрузку на систему или сеть.

- `OnCalendar`
  Задаёт расписание запуска сервисов. Общий формат `ГГГГ-ММ-ДД ЧЧ:ММ:СС`. Примеры использования этой директивы:

Директива	Описание
`OnCalendar=hourly`	Каждый час в начале часа.
`OnCalendar=daily`	Каждый день в полночь.
`OnCalendar=weekly`	Каждую неделю в понедельник 00:00.
`OnCalendar=monthly`	Первого числа каждого месяца в 00:00.
`OnCalendar=yearly`	Первого января каждого года в 00:00.
`OnCalendar=*-*-* 12:00:00`	Каждый день ровно в 12:00.
`OnCalendar=Mon..Fri 09:00`	Каждый будний день в 9:00.
`OnCalendar=Sat,Sun 18:00`	По выходным в 18:00.
`OnCalendar=Mon 10:00, Wed 12:00`	По понедельникам в 10:00 и по средам в 12:00.
`OnCalendar=2025-12-31 23:59:00`	Один раз в указанную дату и время.
`OnCalendar=*:*:10`	Раз в минуту. Каждый раз, когда значение секунд становится равным 10.
`OnCalendar=*:*:0/15`	Каждый раз, когда значение секунд становится кратным 15. То есть четыре раза в минуту (0, 15, 30, 45).
`OnCalendar=*:*:25/10`	Начиная с 25 секунд, с шагом в 10 секунд. Четыре раза в минуту (25, 35, 45, 55).
`OnCalendar=*:20`	Раз в час. Когда значение минут становится равным 20.
`OnCalendar=*:0/20`	Три раза в час. Когда значение минут становится кратным 20.
`OnCalendar=7`	Раз в сутки. Когда значение часов становится равным 7.
`OnCalendar=0/7`	Три раза в сутки. Когда значение часов становится кратным 7.
`OnCalendar=0/7:12:34`	Три раза в сутки. Когда значение часов становится кратным 7, а значение минут становится равным 12 и значение секунд становится равным 34.
`OnCalendar=*-01,04-0/3 2/5:12`	Любой год. Январь или апрель. Значение дня кратно трём. Значение часа равно 2, 7, 12, 17 или 22. Значение минут равно 12, а секунд 0.

Директивы секции `[Path]`

Path-юниты systemd (*.path) запускают соответствующие сервисы (*.service) при некоторых изменениях в файловой системе. Имя path-юнита и сервиса должны совпадать.

- `Unit`
  Задаёт имя юнита, запускаемого этим path-юнитом. По умолчанию path-юнит запускает сервис с тем же именем.
- `PathExists`
  Запускает сервис, когда появится указанный файл или директория.
- `PathExistsGlob`
  Как `PathExists`, но с glob-шаблоном. То есть можно использовать метасимволы `*`, `?`, `[...]`.
- `PathChanged`
  Запускает сервис, на любое изменение файла: запись, удаление, изменение атрибутов, изменения размера.
- `PathModified`
  Запускает сервис, при каждом изменении содержимого файла.
- `DirectoryNotEmpty`
  Запускает сервис, если директория перестаёт быть пустой.

- `MakeDirectory`
  По умолчанию, если вы укажете `PathExists=/some/dir/file` или `DirectoryNotEmpty=/some/dir`, а директория `/some/dir` не существует, systemd просто не сможет отслеживать её состояние. Если добавить `MakeDirectory=true`, то при активации path-юнита systemd создаст эту директорию (с правами по умолчанию 0755 и владельцем `root`), чтобы можно было отслеживать её изменения.

Спецификаторы внутри юнит-файла

Внутри юнит-файлов можно использовать следующие спецификаторы.

Спецификатор	Назначение
`%n`	Имя юнита (например, nginx.service).
`%p`	Имя родительского юнита.
`%i`	Имя инстанса, например, если юнит называется `myservice@first`, то `%i = first`
`%f`	Полный путь к файлу юнита.
`%u`	Имя пользователя, от которого запускается сервис.
`%h`	Домашний каталог пользователя.
`%U`	UID пользователя.
`%m`	Machine ID системы.
`%H`	Имя хоста.