

# Практика управления системой

## Текстовые конфиги

Проблема хранения и редактирования конфигурации в ОС Линукс до сих пор это решается старым, проверенным в `unix`-системах, способом: любой файл настроек --- человекочитаемый текст. Один из вопросов, который возникает в процессе диалога с гуру --- а что такое плоский текст? Когда говорят про текстовый файл, то имеют в виду файл в виде плоского текста (англ. *plain text*). Если текст --- некий поток символов, то плоский текст хранит только собственно информацию, а размеченный текст хранит и некоторую метainформацию о, например, их внешнем виде. Таким образом, когда вы видите содержимое плоского текстового файла, то вы видите именно то, что должны видеть. Размеченный же текст можно показывать в виде плоского текста и в формате представления. Когда говорят, что любой файл является текстовым, то это значит, что он всегда доступен как текст плоский, то есть его можно редактировать текстовым редактором.

В качестве отступления напомним, какие в ОС Линукс бывают текстовые редакторы. Все текстовые редакторы можно разделить на:

- классические редакторы (`vi`, `vim`, `emacs`), которые могут работать как в консоли, так и в графическом окружении;
- современные редакторы, требующие графической среды (например, редактор `kate` для среды `KDE`);
- простые редакторы для эпизодического использования (например, в ПСПО входит простой редактор `mcedit`).

Остается напомнить, что программы типа `OpenOffice Writer` текстовыми редакторами не являются: их принято называть текстовыми процессорами, и работают они с размеченным текстом.

Важным следствием хранения конфигурации в текстовых файлах является возможность использовать стандартный набор утилит для обработки текста, таких как `sed`, `grep`, `tail`, `head`, `echo`. Особенно следует здесь отметить утилиту для автоматической замены в файлах `sed`.

Опишем общие концепции системы настроек в `unix`-подобных системах.

**Пространство имён --- файловая система.** Для того, чтобы настройки одной программы не путались с настройками другой, необходимо использовать пространства имён. В нашем случае пространство имён даёт файловая система: зачем множить сущности (создавать некоторую древовидную классификацию настроек), если дерево каталогов уже есть? В каталоге `/etc` лежат файлы, необходимые для настройки остальной части системы, а пользовательские настройки лежат в каталоге пользователя в файлах и каталогах, имя которых начинается с точки ("`.`", англ. *dot files*), по-умолчанию пользователю их не видно.

**У каждого пакета --- свои файлы конфигурации.** В правильном дистрибутиве ОС Линукс Каждый файл принадлежит какому-то одному (и только одному) пакету. Имена большинства подкаталогов в `/etc` соответствуют названиям пакетов, хотя и далеко не всегда с ними точно совпадают. Последний момент связан с тем, что имена каталогов, например `/etc/ssh`, обычно одинаковы для различных `unix`-подобных систем, а вот имена пакетов определяются внутренней политикой ПСПО и обычно совпадают с названием программы.

**Гибкость представления --- пишите, что хотите.** Это приводит к разнообразию форматов файлов конфигурации: какие-то конфигурации линейны, какие-то --- древовидны, а какие-то

вообще являются сценарием языка shell. За этот "зоопарк" часто упрекают unix-системы, но, с другой стороны, это позволяет выбирать удобный формат под каждую конкретную задачу.

**Чтение и модификация с использованием разнообразных инструментов.** С одной стороны, для ряда задач есть графические конфигураторы программы. В каком-то смысле они удобные, но обычно они делают настройку графического интерфейса и стандартных типовых задач. А каковы должны быть инструменты, которые бы позволили читать и изменять практически любой конфигурационный файл? Поскольку мы договорились, что конфиги представляются в виде текста, то у нас есть следующие инструменты для работы с текстом:

- текстовый редактор (если нет задачи автоматизации изменений);
- утилиты для обработки текста и работы с файлами и язык программирования shell в случае необходимости или желания автоматизировать работу с конфигурацией;
- интерпретаторы языков программирования (awk, perl, python) для более сложных операций с файлами конфигурации;
- наконец, с текстовыми файлами конфигурации может работать специализированная программа на любом языке программирования (к таким относится, например, графический конфигуратор KDE).

Таким образом, в линуксе есть много программ, которые работают с файловой системой и с содержимым файлов, и именно эти программы и являются инструментом, который позволяет выполнять с текстовыми настройками любые манипуляции, в том числе автоматически.

## Уровни выполнения

Расскажем немного об уровнях выполнения системы (англ. *runlevel*). Уровень выполнения представляет собой специальную абстракцию, упоминаемую в файле `/etc/inittab`. Это конфигурационный файл для первого запускаемого при загрузке системы процесса, который носит имя `init`. Изменять данный файл приходится весьма нечасто, причем большинство изменений касаются только строки с параметром `initdefault`: в ней задается уровень выполнения по умолчанию. Файл `/etc/inittab` лишь создает разделение на уровни выполнения: за конфигурацию набора запускаемых на каждом уровне служб в дистрибутивах ПСПО отвечает утилита `chkconfig`. Она манипулирует ссылками на службы, расположенными в каталогах вида `/etc/rc<уровень>.d`. Таким образом, команда

```
$ ls -l /etc/rc0.d/
```

покажет службы, выполнение которых затрагивает переход на уровень 0.

Администратор системы может переключать систему с одного уровня выполнения на другой, выполняя команду `init` с единственным параметром --- номером нужного уровня. При переходе система определяет, какие службы должны быть запущены, а какие --- остановлены, производит собственно переключение и выполняет необходимые операции.

Сами уровни выполнения можно, хотя и с некоторой натяжкой, воспринимать как профили работы системы. Кроме того, к уровням выполнения причисляют также некоторые специальные действия:

- уровень 0 --- полная остановка системы с выключением (если это возможно);
- уровень 6 --- перезагрузка, то есть остановка и загрузка заново (на уровень по умолчанию).

Что касается других уровней, то чаще всего они используются так:

- уровень 1 --- однопользовательский режим работы;
- уровень 2 --- многопользовательский режим без сети;
- уровень 3 --- многопользовательский режим с сетью.

Списка всех уровней это, однако, не исчерпывает. Поведение системы на оставшихся уровнях выполнения устоялось недостаточно, хотя можно отметить используемое еще одно соглашение, используемое во многих системах:

- уровень 5 --- многопользовательский режим с сетью и графическим интерфейсом (однако в ряде дистрибутивов GNU/Linux это уровень 2!).

Отметим, что в дистрибутивах ПСПО дополнительно выполняется еще одно соответствие:

- уровень 7 --- работа инсталлятора системы.

Обратим внимание на то, что организованная таким способом система довольно неудобна. Уровни выполнения уже давно не проходятся подряд: не происходит загрузки на первый уровень с последующим переключением на второй, третий и далее до нужного. Фактически, разрешен непосредственный переход с любого уровня выполнения на любой другой --- иными словами, само слово "уровень" потеряло в данной схеме свое значение. Почему уровней выполнения (по сути дела --- профилей работы системы) существует столько, сколько сейчас --- вопрос традиции, а не удобства. В BSD-системах, например, существуют только два уровня выполнения: однопользовательский и штатный.

Чтобы понять, почему загрузка системы устроена именно так, вспомним некоторые факты из истории. Существующая схема, называемая часто System V init, была написана очень давно, еще во времена коммерческого UNIX. Создатели UNIX --- Кен Томпсон (Ken Thompson) и

Деннис Ритчи (Dennis Ritchie) --- еще ранее работали над операционной системой Multics (совместный проект MIT, GE и Bell Labs, финансируемый Пентагоном). В системе Multics существовала появившаяся по требованию военных схема разделения на "уровни доступа":

- Первый уровень доступа к системе определялся следующим образом. Каждый сеанс доступа к данным управлялся одним и тем же субъектом --- ясно, что с данными этот субъект мог делать все, что ему заблагорассудится. Необходимости защищать данные от произвольного доступа, таким образом, не было, а от непроизвольного --- была: если в системе одновременно функционировали несколько процессов, они не должны были иметь возможности испортить данные друг друга.
- При втором уровне доступа в системе могли работать сразу несколько субъектов одного класса. Эти субъекты составляли "команду" и потому не имели серьезных поражений в правах по сравнению друг с другом. Тем не менее, при такой схеме работал механизм защиты от произвольного доступа, представляющий по сути дела Access Control List (ACL).
- Третий уровень доступа соответствовал работе в системе пользователей разных категорий. Условно говоря, подключенные к машине терминалы могли соответствовать пользователям одной категории, а соединения с системой по сети --- пользователям другой категории. Для каждой из таких категорий алгоритм определения прав доступа мог задаваться отдельно.

Классическая UNIX-система содержала следы этой запутанной политики:

- Первый уровень доступа частично соответствовал однопользовательскому режиму, который давал монопольный доступ к консоли машины.
- Второй уровень доступа частично соответствовал многопользовательскому режиму: сидящие за терминалами пользователи образовывали "команду".
- Третий уровень доступа частично соответствовал подключению машины к сети.

Различные режимы работы в UNIX были названы уровнями выполнения, причем ни о какой гибкости реализации речи не шло. В Multics права доступа на третьем уровне могли различаться даже алгоритмами определения: для одной категории --- ACL, для другой --- мандаты и пр. В UNIX это было сведено к некоторым упрощенным представлениям, уже не зависящим от уровня выполнения: на первое место вышла файловая система, пользователи и группы, причем вычисление прав доступа стало определяться в соответствии с жестким, заранее заданным алгоритмом. Возможность делать с системой все, что угодно, была привязана не к уровню выполнения, а к доверенному субъекту --- пользователю с идентификатором (*uid*), равным нулю.

Заметим, что такая архитектура позволяла организовать следующую схему. На первом уровне можно было запускать некоторые "системные" службы, на втором --- добавлять к ним службы, различающие пользователей, на третьем --- службы, организующие доступ пользователей из сети. Так устоялась традиция разделения на однопользовательский режим, многопользовательский режим и многопользовательский режим с сетью, которые соответствуют 1-му, 2-му и 3-му уровням выполнения. В настоящее время описанная модель потеряла практический смысл (за реализацию различных алгоритмов определения прав доступа может отвечать, к примеру, SELinux), а потому соответствующая схема загрузки служб (называемая *System V init*) стала восприниматься отчасти как наследие прошлого (в английском языке используется термин *legacy*). Тем не менее, в большинстве дистрибутивах GNU/Linux поддерживается однопользовательский режим (первый уровень выполнения), иногда используемый для монопольной работы с системой администратора при помощи консоли. В него можно перейти командой

```
# init 1
```

или указав при загрузке системы параметр ядра "single".