10

Конфигурация: правильное начало

> Стоит внимательно посмотреть на наши истоки, и результаты организуются сами собой.
>
> —Александр Кларк (Alexander Clark)

В операционной системе Unix программы могут обмениваться данными со своим окружением различными способами. Эти способы удобно разделить на (a) опрос параметров среды и (b) интерактивные каналы. В данной главе основное внимание уделено опросу параметров среды. Интерактивные каналы рассматриваются в следующей главе.

10.1. Конфигурируемые параметры

Прежде чем погрузиться в детали различных видов конфигурации программ, следует ответить на важнейший вопрос: “Какие параметры должны быть конфигурируемыми?”

Интуитивно Unix-программист ответит: “Все”. Правило разделения, которое рассматривалось в главе 1, подталкивает Unix-программистов создавать механизм и там, где это возможно, передавать решения о политике на ближайшие к пользователю уровни. Хотя в таком случае часто создаются мощные и полезные для высококвалифицированных пользователей программы, столь же часто создаются интерфейсы, ошеломляющие новичков и случайных пользователей избытком возможных вариантов и конфигурационных файлов.

В ближайшее время Unix-программисты не собираются избавляться от склонности разрабатывать программы для коллег или наиболее опытных пользователей (в главе 20 рассматривается вопрос о том, будут ли такие перемены действительно желаемыми). Поэтому, вероятно, полезнее будет изменить вопрос: какие параметры *не* должны быть конфигурируемыми? В Unix-практике имеются некоторые определяющие принципы для решения данной проблемы.

Прежде всего, *не следует создавать конфигурационные ключи для тех параметров, которые можно надежно определить автоматическим путем.* Такая ошибка поразительно широко распространена. Вместо этого, рекомендуется искать способы устранения конфигурационных ключей путем автоопределения или пытаться использовать альтернативные методы до тех пор, пока один из них не достигнет цели. Если разработчик считает такой подход грубым или слишком дорогим, то ему следует спросить себя, не начал ли он преждевременную оптимизацию.

> Я столкнулся с одним из наилучших примеров автоопределения, когда мы с Деннисом Ритчи переносили Unix на Interdata 8/32. Это была машина с обратным порядком следования байтов. Требовалось генерировать для нее данные на компьютере PDP-11, записывать магнитную ленту и загружать эту ленту в Interdata. Обычно ошибка возникала, когда мы забывали изменить порядок следования байтов. Ошибка контрольной суммы показывала, что необходимо отключить ленту, снова подключить ее к PDP-11, заново создать ленту, отключить ее и подключить опять. Однажды Деннис усовершенствовал программу считывания магнитной ленты на Interdata таким образом, что в случае возникновения ошибки контрольной суммы лента перематывалась, активизировался ключ "byte flip" (преобразование порядка байтов) и программа повторно считывала данные с ленты. Вторая ошибка контрольной суммы прерывала загрузку, но в 99 % случаев осуществлялось повторное считывание ленты и выполнение верных действий. Продуктивность нашей работы мгновенно возросла, и с того времени мы почти полностью игнорировали порядок следования байтов на ленте.
>
> Стив Джонсон.

Существует хорошее эмпирическое правило: необходимо приспосабливаться, если затраты на это не превышают 0,7 сек задержки. 0,7 сек — "магическое" число, поскольку, как обнаружил Джеф Раскин во время разработки Canon Cat, люди почти не способны заметить более короткую задержку при запуске. Она теряется среди ментальных издержек при переключении фокуса внимания.

Второй принцип заключается в том, что *ключи оптимизации должны быть невидимыми для пользователей.* Экономичная работа программы — задача *проектировщика*, а не пользователя. Минимальный прирост производительности, который может получить пользователь с помощью ключей оптимизации, обычно не стоит затрат, связанных со сложностью интерфейса.

> Unix всегда избегала бессмыслицы файловых форматов (таких как длина записи, фактор блокировки и другие), однако подобная бессмыслица вернулась в чрезмерной, вязкой массе конфигурации. Принцип KISS стал принципом MICAHI: make it complicated and hide it (усложнить и скрыть).
>
> Дуг Макилрой.

Наконец, *с помощью конфигурационного ключа не следует делать то, что можно сделать при помощи сценария-упаковщика или тривиального конвейера.* Не следует вносить в программу дополнительную сложность, если для решения задачи можно без труда задействовать другие программы. (В главе 7 показано, почему утилита *ls(1)* не содержит встроенного средства формирования страниц или параметра для вызова такого средства.)

Существует несколько более общих вопросов, которые следует обдумывать всякий раз при появлении желания добавить конфигурационный параметр.

- Можно ли исключить данную функцию? Почему "раздувается" справочное руководство, а пользователь перегружен излишними подробностями?
- Можно ли безопасно изменить обычный режим работы программы так, чтобы данный параметр был необязательным?
- Является ли данный параметр только косметическим? Возможно, следует меньше думать о конфигурируемости пользовательского интерфейса и больше внимания уделять корректности его работы?
- Не следует ли сделать режим, активизированный данным параметром, в виде отдельной программы?

Увеличение числа излишних параметров чревато множеством негативных последствий. Одно из наименее очевидных и самых серьезных — это влияние на тестовое покрытие.

> Добавление конфигурационного параметра on/off, в случае если оно не сделано очень тщательно, может привести к необходимости удвоения количества тестов. Поскольку на практике количество тестов никогда не удваивается, практический эффект заключается в сокращении количества тестов для любой заданной конфигурации. Десять параметров приводят к 1024 тестированиям, и довольно скоро возникнут реальные проблемы надежности (Стив Джонсон).

10.2. Месторасположение конфигурационной информации

Классическая Unix-программа может искать управляющую информацию в 5 группах параметров начальной загрузки:

- файлы конфигурации в каталоге `/etc` (или в другом постоянном каталоге системы);
- системные переменные окружения;
- файлы конфигурации (или файлы профилей (dotfile)) в начальном каталоге пользователя (описание данной важной концепции приведено в главе 3);
- пользовательские переменные окружения;
- ключи и аргументы, переданные программе в командной строке во время вызова.

Данные запросы обычно выполняются в описанном выше порядке. Поэтому более поздние (локальные) установки заменяют более ранние (глобальные). Установки, найденные раньше, могут способствовать программе в определении области для дальнейшего поиска конфигурационных данных.

При выборе механизма для передачи программе конфигурационных данных необходимо помнить о том, что хорошая Unix-практика требует использования того механизма, который наиболее близко соответствует ожидаемому времени жизни настроек. Следовательно: для настроек, которые, весьма вероятно, изменяются между вызовами программы, следует использовать ключи командной строки. Для предпочтений, которые изменяются редко, но должны находиться под индивидуальным контролем пользователя, следует использовать конфигурационный файл в начальном каталоге данного пользователя. Для хранения информации о настройках, которые должны устанавливаться системным администратором для всего узла и *не* изменяются пользователями, используется конфигурационный файл в системной области.

Ниже каждая область конфигурационных параметров рассматривается более подробно, после чего приводится несколько учебных примеров.

10.3. Файлы конфигурации

Файл конфигурации (run-control file) — файл объявлений или команд, связанных с программой, которая интерпретирует его во время запуска. Если программа имеет специфическую для данного узла конфигурацию, которая совместно используется всеми пользователями узла, то часто файл конфигурации находится в каталоге `/etc`. (В некоторых Unix-системах для хранения таких данных имеется подкаталог `/etc/conf`.)

Сведения пользовательской конфигурации часто находятся в скрытом конфигурационном файле в начальном каталоге данного пользователя. Такие файлы часто называются “dotfiles” (файлы профилей), поскольку они используют Unix-соглашение о том, что имя файла, начинающееся с точки, невидимо для средств создания перечня файлов каталога[1].

Программы также могут иметь конфигурационные или скрытые каталоги. В таких каталогах собираются группы конфигурационных файлов, которые относятся к программе, но их удобнее интерпретировать отдельно (возможно потому, что они связаны с различными подсистемами программы или имеют разный синтаксис).

В настоящее время, независимо от того, используется файл или каталог, соглашение требует, чтобы месторасположение конфигурационной информации имело то же базовое имя что и считывающий ее исполняемый файл. В более раннем соглашении, которое до сих пор широко распространено в системных программах, используется имя исполняемого файла с суффиксом “rc” (от “run control” — управление работой)[2]. Поэтому опытный пользователь Unix, работая с программой “seekstuff”, имеющей как общесистемную, так и пользовательскую конфигурацию, ожидает найти первую в файле `/etc/seekstuff`, а последнюю — в файле `.seekstuff` в своем начальном каталоге. Вместе с тем он не удивился бы, обнаружив файлы `/etc/seekstuffrc` и `.seekstuffrc`, особенно если программа seekstuff является какой-либо системной утилитой.

---

1 Для отображения скрытых файлов используется параметр `-a` утилиты *ls(1)*.

2 Суффикс “rc” связан с системой, предшествующей Unix, CTSS. В ней присутствовала функция сценария команд, которая называлась “runcom”. В ранних Unix-системах имя “rc” использовалось для загрузочного сценария операционной системы как дань runcom в CTSS.

В главе 5 описывался несколько отличающийся набор правил проектирования текстовых файловых форматов, а также рассматривались способы оптимизации по различным критериям способности к взаимодействию, прозрачности и экономичности транзакций. Обычно файлы конфигурации только считываются однажды во время запуска программы и не перезаписываются. Следовательно, экономия обычно не является основной проблемой. Возможность взаимодействия и прозрачность подталкивают разработчиков к использованию текстовых форматов, разработанных для чтения людьми и редактирования с помощью обычного текстового редактора.

Несмотря на то, что семантика конфигурационных файлов, несомненно, полностью зависит от программы, существует несколько широко распространенных правил проектирования, связанных с конфигурационным синтаксисом. Они будут описаны далее, а до этого следует рассмотреть важное исключение.

Если программа является интерпретатором какого-либо языка, то ожидается, что существует просто файл команд в синтаксисе данного языка, которые выполняются во время запуска программы. Данное правило важно потому, что Unix-традиции твердо поддерживают разработку всех видов программ как языков специального назначения и мини-языков. Широко известные примеры с файлами профилей такого типа включают в себя различные командные оболочки Unix и программируемый редактор *Emacs*.

Основанием для данного правила проектирования является убеждение, что частные случаи — негативное явление. Поэтому, любой ключ, изменяющий поведение языка, должен устанавливаться изнутри данного языка. Если при разработке языка обнаруживается, что выразить в самом языке все его начальные установки *невозможно*, то Unix-программист сказал бы, что в конструкции имеется проблема, которую, вероятнее всего, необходимо исправить, а не разрабатывать конфигурационный синтаксис частного случая.

После рассмотрения исключения следует описать правила обычного стиля конфигурационного синтаксиса. Исторически сложилось так, что моделью для них послужил синтаксис оболочек в Unix.

1. *Поддержка объясняющих комментариев, начинающихся с символа* #. Синтаксис также должен игнорировать пробел перед символом #, для того чтобы обеспечить поддержку комментариев в строках, содержащих конфигурационные директивы.
2. *Предотвращение опасных различий в пустом пространстве.* То есть последовательности пробелов и символов табуляции должны синтаксически интерпретироваться как один пробел. Если формат директив ориентирован на строки, то следует игнорировать завершающие пробелы и символы табуляции в строках. Метаправило заключается в том, что интерпретация файла не должна зависеть от невидимых для человеческого глаза различий.
3. *Интерпретация нескольких пустых строк и строк комментариев как одной пустой строки.* Если во входном формате в качестве разделителей записей используются пустые строки, то, возможно, понадобится гарантия того, что строка комментария не завершает запись.
4. *Лексическая интерпретация файла как простой последовательности лексем, разделенных пробелами, или строк лексем.* Человеку трудно изучить, запомнить и проанализировать сложные лексические правила, поэтому их следует избегать.

5. *Однако следует поддерживать строковый синтаксис для лексем, содержащих пробелы.* В качестве сбалансированных разделителей можно использовать одинарные или двойные кавычки. Если поддерживаются оба вида кавычек, то не следует назначать им различную семантику, что характерно для shell, поскольку это является широко известным источником путаницы.
6. *Поддержка синтаксиса обратной косой черты для внедрения в строки непечатаемых и специальных символов.* Стандартным образцом в данном случае является синтаксис escape-последовательностей, поддерживаемый компиляторами C. Поэтому, например, было бы весьма неожиданно, если бы строка `"a\tb"` интерпретировалась бы иначе, чем символ “a”, за которым следует символ табуляции, за которым в свою очередь следует символ “b”.

С другой стороны, в конфигурационном синтаксисе *не* следует копировать некоторые аспекты shell-синтаксиса — по крайней мере, не имея значительной и специфической причины. В данную категорию попадают “вычурные” правила использования кавычек и скобок в shell, а также специальные метасимволы shell для масок и подстановки переменных.

Целью данных соглашений является сокращение количества нововведений, с которыми приходится справляться пользователям при чтении и редактировании конфигурационного файла для новой программы. Следовательно, если понадобилось нарушить данные соглашения, то необходимо попытаться сделать необходимость нарушений очевидной, а, кроме того, документировать синтаксис с особенной тщательностью, и (что наиболее важно) проектировать так, чтобы его можно было легко понять из примеров.

Данные правила стандартного стиля только описывают соглашения о лексемах и комментариях. Имена конфигурационных файлов, их высокоуровневый синтаксис и семантическая интерпретация синтаксиса обычно отражают специфику приложения. Однако существует несколько исключений из данного правила; одно из них — файлы профилей, которые стали “широко известными” в том смысле, что обычно содержат информацию, используемую целым классом приложений. Такое совместное использование форматов конфигурационных файлов сокращает количество нововведений, в которых приходится разбираться пользователям.

Среди таких файлов наиболее “укоренившимся”, вероятно, является файл `.netrc`. Клиентские Internet-программы, которые должны отслеживать пары узел/пароль для пользователя, обычно могут получать их из файла `.netrc`, если таковой существует.

10.3.1. Учебный пример: файл `.netrc`

Файл `.netrc` — хороший пример стандартных правил в действии. Пример 10.1 содержит пароли, измененные в целях защиты реального пользователя.

Следует отметить, что разобраться в данном формате не сложно, даже встретив его впервые. Он представляет собой набор троек машина/имя/пароль, каждая из которых описывает учетную запись на удаленном узле. Такая прозрачность фактически гораздо более важна, чем экономия времени путем быстрой интерпретации

или экономия пространства путем использования более компактного и непонятного файлового формата. Он позволяет сэкономить гораздо более ценный ресурс — *человеческое* время, поскольку увеличивает вероятность того, что человек сможет прочесть и модифицировать данный формат без изучения руководства или использования инструмента, менее знакомого, чем простой текстовый редактор.

Пример 10.1. Файл `.netrc`

```
# FTP-доступ к моему Web-узлу
machine Unix1.netaxs.com
        login esr
        password joesatriani

# Мой главный почтовый сервер в Netaxs
machine imap.netaxs.com
        login esr
        password jeffbeck

# Дополнительный почтовый ящик IMAP в CCIL
machine imap.ccil.org
    login esr
    password marcbonilla

# Дополнительный почтовый ящик POP в CCIL
machine pop3.ccil.org
    login esr
    password ericjohnson

# Учетная запись shell в CCIL
machine locke.ccil.org
    login esr
    password stevemorse
```

Также следует отметить то, что данный формат используется для предоставления информации нескольким службам. Это является преимуществом, поскольку означает, что конфиденциальную информацию о паролях необходимо хранить только в одном месте. Формат `.netrc` был разработан для первоначальной клиентской FTP-программы в Unix. Данный формат используется всеми FTP-клиентами, а также распознается некоторыми telnet-клиентами, CLI-инструментом *smbclient(1)* и программой *fetchmail*. При разработке Internet-клиента, который должен выполнять аутентификацию паролей посредством удаленной регистрации, правило наименьшей неожиданности требует, чтобы он по умолчанию использовал содержимое файла `.netrc`.

10.3.2. Переносимость на другие операционные системы

Общесистемные конфигурационные файлы являются конструкторской тактикой, которую можно применять почти во всех операционных системах, однако довольно трудно найти соответствие файлам профилей в не-Unix-окружении. Критическим элементом, недостающим в большинстве операционных систем, отличных от

Unix, является действительная поддержка многопользовательской работы и понятие начального каталога для каждого пользователя. В операционной системе DOS и версиях Windows вплоть до ME (включая Windows 95 и 98), например, такая идея полностью отсутствует; вся конфигурационная информация должна храниться либо в общесистемных файлах конфигурации в фиксированной области, реестре Windows, либо в том же каталоге, из которого запускается программа. В Windows NT имеется некоторое понятие о начальных каталогах пользователей (которое послужило началом пути к Windows 2000 и XP), но оно только недостаточно поддерживается инструментальными средствами системы.

10.4. Переменные окружения

Когда запускается какая-либо Unix-программа, доступная ей среда включает в себя набор связей "имя–значение" (как имена, так и значения являются строками). Некоторые из них устанавливаются пользователем вручную, другие — системой во время регистрации пользователя, либо оболочкой или эмулятором терминала (если таковой используется). В операционной системе Unix в переменных окружения хранится информация о путях поиска файлов, системных установках по умолчанию, номер процесса и идентификатор текущего пользователя, а также другие сведения о среде выполнения программ. Команда **`set`**, введенная в приглашении shell, отображает полный список определенных в текущий момент переменных оболочки.

В языках C и C++ значения данных переменных запрашиваются с помощью библиотечной функции *getenv(3)*. В языках Perl и Python во время запуска инициализируются объекты словарей среды. В других языках, как правило, используется одна из двух перечисленных моделей.

10.4.1. Системные переменные окружения

Существует множество широко известных переменных окружения, значения которых программа может получить при запуске из оболочки Unix. Данные переменные (особенно `HOME`) часто требуется оценить *до* считывания локального файла профиля.

`USER`
: Регистрационное имя учетной записи, под которым инициируется данный сеанс (BSD-соглашение).

`LOGNAME`
: Регистрационное имя учетной записи, под которым инициируется данный сеанс (соглашение System V).

`HOME`
: Начальный каталог пользователя, инициализирующего данный сеанс.

`COLUMNS`
: Количество символьных столбцов управляющего терминала или окна эмулятора терминала.

LINES
: Количество символьных строк управляющего терминала или окна эмулятора терминала.

SHELL
: Имя командной оболочки данного пользователя (часто используется командами, создающими подоболочку).

PATH
: Список каталогов, которые просматривает оболочка при поиске исполняемых команд, соответствующих заданному имени.

TERM
: Тип терминала для консоли сеанса или окна эмулятора терминала (предварительные сведения представлены в учебном примере terminfo в главе 6). TERM — переменная, специально используемая в таких программах для создания удаленных сеансов через сеть (таких как *telnet* и *ssh*), предполагается, что они передают и устанавливают значение данной переменной в удаленном сеансе.

(Данный список характерен для Unix-систем, однако он не является исчерпывающим.)

Переменная `HOME` особенно важна, поскольку многие программы используют ее для поиска файлов профилей вызывающего пользователя (другие программы вызывают некоторые функции в динамической C-библиотеке для выяснения начального каталога вызывающего пользователя).

Следует отметить то, что некоторые или все данные системные переменные окружения могут *не* устанавливаться во время запуска программы, если программа запускается методом, отличным от создания подпроцесса в shell. В частности, демоны-слушатели какого-либо TCP/IP-сокета часто не имеют таких установленных переменных, а если имеют, то их значения едва ли будут полезны.

Наконец, следует отметить, что существует традиция (проиллюстрированная переменной `PATH`) использования двоеточия в качестве разделителя, когда переменная окружения должна содержать несколько полей, особенно если данные поля можно интерпретировать как некоторые пути поиска. Заметим, что некоторые оболочки (особенно `bash` и `ksh`) *всегда* интерпретируют разделенные двоеточиями поля в переменной окружения как имена файлов. Это, в частности, означает, что символ ~ в таких полях преобразовывается в путь к начальному каталогу данного пользователя.

10.4.2. Пользовательские переменные окружения

Несмотря на то, что приложения могут свободно интерпретировать переменные окружения за пределами определенного системой набора, в настоящее время фактическое использование такой возможности является довольно необычным. Значения переменных окружения в действительности непригодны для передачи структурированной информации в программу (хотя в принципе это реализуемо посредством синтаксического анализа значений). Вместо этого в современных Unix-приложениях используются конфигурационные файлы, а также файлы профилей.

Существует, однако, несколько конструкторских моделей, в которых могут оказаться полезными переменные окружения, определяемые пользователем.

Независимые от приложения настройки, которые должны совместно использоваться большим количеством различных программ. Данный набор "стандартных" настроек изменяется исключительно медленно, поскольку множеству программ требуется опознать каждую настройку до того, как она станет полезной[3]. Ниже приводятся стандартные переменные.

EDITOR

Имя предпочтительного для пользователя редактора (часто используется командами, создающими подоболочку)[4].

MAILER

Имя предпочтительного для пользователя почтового агента (часто используется командами, создающими подоболочку).

PAGER

Имя предпочитаемой пользователем программы для просмотра простого текста.

BROWSER

Имя предпочитаемой пользователем программы для просмотра информации в Web. Данная переменная до сих пор считается новой и еще широко не распространена.

10.4.3. Когда использовать переменные окружения

Общим для пользовательских и системных переменных окружения является то обстоятельство, что в них содержатся данные, хранение которых в большом количестве конфигурационных файлов было бы утомительным. И крайне утомительным было бы изменение данной информации во всех файлах конфигурации при изменении настроек. Обычно пользователи задают значения данных переменных в своих файлах начальной конфигурации shell-сеанса.

Значение отличается в нескольких совместно использующих файл профиля контекстах, или родительский процесс должен передать информацию множеству дочерних процессов. Ожидается, что некоторые блоки конфигурационной информации будут отличаться между несколькими контекстами, в которых вызывающий пользователь совместно использовал бы общие файлы конфигурации и файлы профиля. В качестве примера системного уровня можно рассмотреть несколько shell-сеансов, открытых через окна эмулятора терминала на рабочем столе системы X. Все они считывают одни и те же файлы профилей, но могут иметь различные значения пере-

---

3 Никто не знает действительно изящного способа представить эти распределенные данные о настройках. Переменные среды, вероятно, не являются этим способом, однако для всех известных альтернатив характерны одинаково неприятные проблемы.

4 В действительности, большинство Unix-программ вначале проверяют переменную `VISUAL`, и только если она не установлена, обращаются к переменной `EDITOR`. Это пережиток того времени, когда пользователи имели различные настройки для строковых и визуальных редакторов.

менных COLUMNS, LINES и TERM. (Данный метод широко использовался в shell-программировании старой школы, а в make-файлах используется до сих пор.)

Значение изменяется слишком часто для файлов профилей, но не при каждом запуске. Определяемая пользователем переменная окружения может (например) использоваться для передачи расположения файловой системы или Internet-ресурса, являющегося корнем дерева файлов, с которыми должна работать программа. Так, например, система контроля версий CVS интерпретирует переменную CVSROOT. Несколько клиентских программ чтения новостей, доставляющих новости от серверов с помощью протокола NNTP, интерпретируют переменную NNTPSERVER как расположение запрашиваемого сервера.

Уникальное для процесса переназначение параметров необходимо выразить таким образом, чтобы не требовалось изменять командную строку вызова. Определяемая пользователем переменная среды может быть полезна в ситуациях, когда, по какой-либо причине, может быть неудобно изменять файл профиля приложения или задавать параметры командной строки (возможно, ожидается, что приложение обычно будет использоваться внутри shell-упаковщика или make-файла). Особенно важным контекстом для такого использования является отладка. Например, в Linux использование переменной LD_LIBRARY_PATH, связанной с компонующим загрузчиком *ld(1)*, позволяет изменить место загрузки библиотек — возможно, для выбора версий, которые выполняют проверку переполнения буфера или профилирование (profiling).

Как правило, определяемая пользователем переменная среды может быть эффективным конструкторским выбором, когда значение изменяется настолько часто, что редактирование файла профиля становится неудобным, но не обязательно при каждом вызове (всегда устанавливать расположение с помощью параметра командной строки также было бы неудобно). Как правило, значения таких переменных следует определять *после* локального файла профиля и позволить им переназначать значения файла профиля.

В Unix существует одна традиционная конструкторская модель, которую не рекомендуется применять для новых программ. Иногда пользовательские переменные окружения используются как легковесная замена для выражения настроек программы в файле конфигурации. Например, старая игра *nethack(1)* для того, чтобы получить пользовательские настройки, считывала переменную окружения NETHACKOPTIONS. Такой подход характерен для старой школы. Современная практика в аналогичном случае склонялась бы к синтаксическому анализу настроек из конфигурационного файла .nethack или .nethackrc.

Проблема более раннего стиля заключается в том, что отслеживание месторасположения информации о настройках становится сложнее, чем это было бы, если бы пользователь знал, что в его начальном каталоге находится конфигурационный файл программы. Переменные среды могут быть установлены в любом из нескольких конфигурационных файлов оболочки. В операционной системе Linux в их число, вероятнее всего, входят, как минимум, файлы .profile, .bash_profile и .bashrc. Данные файлы запутаны и ненадежны, поэтому, как только издержки кода, содержащего синтаксический анализатор параметров, стали казаться менее значительными, возникла тенденция к перемещению информации о настройках из переменных окружения в файлы профилей.

10.4.4. Переносимость на другие операционные системы

Переменные среды характеризуются исключительно ограниченной переносимостью из Unix. В операционных системах Microsoft имеются переменные окружения, смоделированные наподобие переменных окружения в Unix, а переменная `PATH` используется, как и в Unix, для установки путей поиска исполняемых файлов, однако большинство других переменных, которые принимаются Unix-программистами как должное (такие как идентификатор процесса или текущий рабочий каталог) не поддерживаются. В других операционных системах (включая классическую MacOS), как правило, отсутствует какой-либо локальный эквивалент переменным окружения.

10.5. Параметры командной строки

Unix-традиции поощряют использование ключей командной строки для управления программами, так чтобы параметры можно было задавать из сценариев. Это особенно важно для программ, которые выполняют функции фильтров или каналов. Существует 3 соглашения, которые определяют способ отделения параметров командной строки от обычных аргументов: исходный стиль Unix, стиль GNU и стиль инструментария X.

В оригинальной традиции Unix для обозначения параметров командной строки служат отдельные буквы с предшествующим дефисом. Параметры режимов, которые не принимают последующих аргументов, можно группировать, поэтому если `-a` и `-b` являются параметрами режима, то использование последовательностей `-ab` или `-ba` также является корректным и активизирует оба режима. Аргумент для параметра, если он существует, указывается после параметра (и отделяется от него пробелом). В данном стиле использование нижнего регистра для обозначения параметров более предпочтительно. Использование в разрабатываемой программе обозначений в верхнем регистре хорошо в том случае, если они представляют собой специальные варианты параметров, обозначаемых буквами нижнего регистра.

Исходный стиль Unix развивался на медленных телетайпах ASR-33, в которых краткость ввода была достоинством; отсюда и использование однобуквенных параметров. Удержание клавиши shift требовало определенных усилий, отсюда предпочтение для нижнего регистра и использование для активизации параметров символа "-" (вместо знака "+", возможно, более логичного в данном случае).

В GNU-стиле для обозначения параметров используются ключевые слова (вместо отдельных букв), которым предшествуют два дефиса. Данный стиль развивался гораздо позднее, когда некоторые из довольно сложных GNU-утилит начали сталкиваться с нехваткой однобуквенных ключей (это было "лечением симптома", но не избавляло от лежащей в основе проблемы). Данный стиль остается популярным, поскольку GNU-параметры читать проще, чем "алфавитную смесь" более ранних стилей. Параметры в GNU-стиле не могут быть сгруппированы без разделяющего пробела. Аргумент параметра (если есть) может отделяться либо пробелом, либо одиночным знаком равенства ("=").

Двойной дефис в GNU-стиле был выбран для того, чтобы в одной командной строке можно было недвусмысленно смешивать традиционные однобуквенные параметры и ключевые слова GNU-стиля. Таким образом, если в первоначально разрабатываемой конструкции присутствует только несколько простых параметров, то можно использовать Unix-стиль, не опасаясь "дня флага", в случае если потребуется в дальнейшем перейти на GNU-стиль. С другой стороны, если используется GNU-стиль, то хорошей практикой будет поддержка однобуквенных эквивалентов, по крайней мере, для наиболее широко используемых параметров.

В стиле X-инструментария для обозначения параметра несколько запутанно используется одиночный дефис и ключевое слово. Данный стиль интерпретируется X-инструментариями, которые фильтруют и обрабатывают определенные параметры (такие как `-geometry` и `-display`) до передачи преобразованной командной строки логике приложения для интерпретации. Стиль X-инструментария не вполне совместим с классическим стилем Unix или GNU-стилем, и его не следует использовать в новых программах, если совместимость с более ранними X-соглашениями не очень важна.

Многие инструментальные средства принимают один дефис, не связанный с какой-либо буквой параметра, как имя псевдофайла, заставляющее приложение считывать данные со стандартного ввода. Кроме того, двойной дефис традиционно распознается как сигнал прекратить интерпретацию параметров и начать буквальную интерпретацию всех последующих аргументов.

Большинство языков программирования в Unix предоставляют библиотеки, которые производят синтаксический анализ командной строки либо в классическом, либо в GNU-стиле (также интерпретируя соглашение по двойному дефису).

10.5.1. Параметры командной строки от `-a` до `-z`

Со временем часто используемые параметры в широко известных Unix-программах создали неформальный стандарт семантики для ожидаемого значения различных флагов. Ниже приводится перечень параметров и их значений, которые будут удобными и привычными для опытных пользователей операционной системы Unix.

`-a`

Все (без аргументов). В GNU-стиле параметр `--all`. Было бы очень неожиданно встретить другое значение для `-a`, кроме синонима `--all`. Примеры: *fuser(1)*, *fetchmail(1)*.

Добавить в конец, как в утилите *tar(1)*. Часто образовывает пару с ключом `-d` для удаления.

`-b`

Размер буфера или блока (с аргументом). Устанавливает размер буфера или размер блока (в программе, осуществляющей архивирование или управление устройствами хранения информации). Примеры: *du(1)*, *df(1)*, *tar(1)*.

Неинтерактивный (пакетный (batch)) режим. Если программа изначально интерактивна, то ключ `-b` может использоваться для подавления подсказок или установки других параметров, необходимых для принятия входных данных из файла, а не от пользователя. Пример: *flex(1)*.

-c

Команда (с аргументом). Если программа является интерпретатором, обычно принимающим команды со стандартного ввода, то ожидается, что аргумент `-c` будет передан программе как одна строка ввода. Данное соглашение особенно четко соблюдается в оболочках и shell-подобных интерпретаторах. Примеры: *sh(1)*, *ash(1)*, *bsh(1)*, *ksh(1)*, *python(1)*. Сравните с ключом `-e` ниже.

Проверка (check) (без аргумента). Проверяется корректность аргумента (или аргументов) файла для команды, но обычная обработка фактически не выполняется. Часто используется как параметр проверки синтаксиса в программах, которые действительно интерпретируют командные файлы. Примеры: *getty(1)*, *perl(1)*.

-d

Отладка (debug) (с аргументом или без). Устанавливает уровень отладочных сообщений. Данное значение весьма распространено.

Иногда ключ `-d` имеет значение "delete" (удалить) или "directory" (каталог).

-D

Определить (define) (с аргументом). Устанавливает значение какого-либо символа в интерпретаторе, компиляторе или (особенно) в приложении с функциями макропроцессора. Моделью послужило использование ключа `-D` в препроцессоре макрокоманд компилятора C. Это устойчивая ассоциация для большинства Unix-программистов; ее не следует нарушать.

-e

Выполнить (execute) (с аргументом). Программы, которые являются упаковщиками или могут использоваться как упаковщики, часто допускают ключ `-e` для определения программы, которой они передают управление.

Редактирование (edit). Программа, способная открыть ресурс либо в режиме только для чтения, либо в режиме редактирования, может принимать ключ `-e` для активизации режима редактирования. Примеры: *crontab(1)* и утилита *get(1)* системы контроля версий SCCS.

Иногда ключ `-e` имеет значение "exclude" (исключить) или "expression" (выражение).

-f

Файл (с аргументом). Очень часто используется с аргументом для указания файла входных (или реже выходных) данных для программ, которым требуется произвольный доступ к входным или выходным данным (там, где недостаточно перенаправления посредством символов < или >). Классическим примером является утилита *tar(1)*; существует также множество других примеров. Кроме того, данный ключ часто используется для указания того, что аргумент, обычно принимаемый из командной строки, необходимо взять из файла. Классические примеры: *awk(1)* и *egrep(1)*. Сравните с ключом `-o` ниже. Часто `-f` представляет собой входной аналог `-o`.

Форсировать (force) (обычно без аргумента). Форсирует некоторые операции (такие как блокировка или разблокировка файлов), которые обычно осуществляются по условию. Данное значение менее распространено.

Демоны часто используют ключ `-f`, комбинируя оба значения, для того чтобы форсировать обработку конфигурационного файла, расположенного в нестандартном месте. Примеры: *ssh(1)*, *httpd(1)* и многие другие демоны.

`-h`

Заголовки (headers) (обычно без аргумента). Включает, подавляет или модифицирует заголовки табличных отчетов, сгенерированных программой. Примеры: *pr(1)*, *ps(1)*.

Помощь (help). В настоящее время данное значение распространено меньше, чем можно ожидать. В большинстве случаев разработчики ранней Unix считали справку лишней тратой памяти, которую они не могли себе позволить. Вместо этого они создавали страницы руководства (что определенным способом сформировало соответствующий стиль справочных руководств в Unix, см. главу 18).

`-i`

Инициализировать (initialize) (обычно без аргумента). Устанавливает некоторый критический ресурс или базу данных, связанную с программой, в первоначальное или пустое состояние. Пример: *ci(1)* в RCS.

Диалоговый (интерактивный (interactive)) режим (обычно без аргумента). Заставляет программу (которая обычно этого не делает) запрашивать подтверждение выполняемых операций. Существуют классические примеры (*rm(1)*, *mv(1)*), однако данное значение не является общепринятым.

`-I`

Включить (include) (с аргументом). Добавляет имя файла или каталога в список просматриваемых приложением ресурсов. Все Unix-компиляторы с любым эквивалентом включения файла исходного кода в своих языках используют ключ `-I` именно в этом смысле. Было бы крайне неожиданно увидеть иное использование данного параметра.

`-k`

Хранить (keep) (с аргументом). Подавляет обычное удаление какого-либо файла, сообщения или ресурса. Примеры: *passwd(1)*, *bzip(1)* и *fetchmail(1)*.

Иногда ключ `-k` имеет значение “kill” (уничтожить, завершить).

`-l`

Вывести список (list) (без аргумента). Если разрабатываемая программа является архиватором или интерпретатором/программой просмотра для какого-либо каталога или архива, то было бы очень неожиданно использовать ключ `-l` для действий, отличных от запроса списка объектов. Примеры: *arc(1)*, *binhex(1)*, *unzip(1)*. (Утилиты *tar(1)* и *cpio(1)* являются исключениями.)

В программах, которые уже являются генераторами отчетов, ключ `-l` почти без исключений означает “long” (длинный) и отображает список в длинном формате, где открываются более подробные сведения по сравнению со стандартным режимом. Примеры: *ls(1)*, *ps(1)*.

Загрузить (load) (с аргументом). Если программа является компоновщиком или интерпретатором языка, то ключ `-l` неизменно загружает библиотеку в некотором соответствующем состоянии. Примеры: *gcc(1)*, *f77(1)*, *emacs(1)*.

Регистрация в системе (login). В таких программах, как *rlogin(1)* и *ssh(1)*, в которых необходимо указать сеть, это можно сделать с помощью ключа `-l`.

Иногда `-l` имеет значение "length" (длина, длительность) или "lock" (блокировка).

`-m`

Сообщение (message) (с аргументом). Ключ `-m` передает свой аргумент в программу как строку сообщения для протоколирования или извещения. Примеры: *ci(1)*, *cvs(1)*.

Иногда ключ `-m` имеет значение "mail" (почта), "mode" (режим) или "modification-time" (время изменения).

`-n`

Число (number) (с аргументом). Используется, например, для указания диапазонов номеров страниц в таких программах, как *head(1)*, *tail(1)*, *nroff(1)* и *troff(1)*. Некоторые сетевые инструменты, обычно отображающие DNS-имена, принимают ключ `-n` как параметр, который вызывает отображение IP-адресов вместо имен. Основные примеры: *ifconfig(1)* и *tcpdump(1)*.

Нет (not) (без аргумента). Используется для подавления обычных действий в таких программах, как *make(1)*.

`-o`

Вывод (output) (с аргументами). Используется для того, чтобы указать программе в командной строке выходной файл или устройство по имени. Примеры: *as(1)*, *cc(1)*, *sort(1)*. Во всех программах с интерфейсами, подобными интерфейсу компилятора, было бы крайне неожиданно увидеть иное использование данного параметра. Программы, поддерживающие ключ `-o` часто (как gcc) имеют логику, которая позволяет опознавать его как после обычных аргументов, так и перед ними.

`-p`

Порт (port) (с аргументом). Главным образом применяется для параметров, которые определяют номера TCP/IP-портов. Примеры: *cvs(1)*, инструменты PostgreSQL, *smbclient(1)*, *snmpd(1)*, *ssh(1)*.

Протокол (protocol) (с аргументом). Примеры: *fetchmail(1)*, *snmpnetstat(1)*.

`-q`

Подавление вывода (quiet) (обычно без аргумента). Подавляет обычное отображение результата или сведений диагностики. Данное значение является весьма распространенным. Примеры: *ci(1)*, *co(1)*, *make(1)*. См. также значение "подавление вывода" ключа `-s`.

`-r` (а также `-R`)

Рекурсия (recurse) (без аргумента). Если программа выполняет операции над каталогом, данный параметр может указывать на то, что операции рекурсивно повторяются для всех входящих в данный каталог подкаталогов. Любое другое применение данного ключа в утилите, обрабатывающей каталоги, было бы весьма неожиданным. Классическим примером, несомненно, является утилита *cp(1)*.

Реверс (reverse) (без аргумента). Примеры: *ls(1)*, *sort(1)*. Данный параметр может использоваться в фильтре для реверсирования своего обычного действия преобразования (сравните с ключом `-d`).

`-s`

Подавление вывода (silent) (без аргументов). Подавляет обычное отображение сведений диагностики или результатов (аналогичен ключу `-q`; когда поддерживаются оба ключа, то `q` означает "quiet" (тихо), а `-s` — "utterly silent" (очень тихо)). Примеры: *csplit(1)*, *ex(1)*, *fetchmail(1)*.

Тема сообщения (subject) (с аргументом). *Всегда* используется в данном значении в командах, которые отправляют или манипулируют почтовыми сообщениями или новостями. Крайне важно обеспечить поддержку данного параметра, поскольку он ожидается программами, отправляющими почту. Примеры: *mail(1)*, *elm(1)*, *mutt(1)*.

Иногда ключ `-s` имеет значение "size" (размер).

`-t`

Тег (tag) (с аргументом). Назвать расположение или передать программе строку, используемую в качестве ключа поиска. Главным образом используется с текстовыми редакторами и программами просмотра. Примеры: *cvs(1)*, *ex(1)*, *less(1)*, *vi(1)*.

`-u`

Пользователь (user) (с аргументом). Указать пользователя по имени или числовому идентификатору (UID). Примеры: *crontab(1)*, *emacs(1)*, *fetchmail(1)*, *fuser(1)*, *ps(1)*.

`-v`

Вывод подробных сведений (verbose) (с аргументом или без). Используется для включения мониторинга транзакций, более объемных списков или вывода отладочных сообщений. Примеры: *cat(1)*, *cp(1)*, *flex(1)*, *tar(1)* и многие другие.

Версия (version) (без аргумента). Отображает версию программы на стандартном выводе и завершает свою работу. Примеры: *cvs(1)*, *chattr(1)*, *patch(1)*, *uucp(1)*. В более распространенном варианте данное действие вызывается ключом `-V`.

`-V`

Версия (version) (без аргумента). Отображает версию программы на стандартный вывод и завершает работу (кроме того, часто распечатывает на экране детали конфигурации, с которой была скомпилирована программа). Примеры: *gcc(1)*, *flex(1)*, *hostname(1)* и многие другие. Было бы крайне неожиданно, если бы данный ключ использовался иным способом.

`-w`

Ширина (width) (с аргументом). Главным образом используется для определения ширины в форматах вывода. Примеры: *faces(1)*, *grops(1)*, *od(1)*, *pr(1)*, *shar(1)*.

Предупреждение (warning) (без аргументов). Включает диагностические предупреждения или подавляет их. Примеры: *fetchmail(1)*, *flex(1)*, *nsgmls(1)*.

-x

Разрешить отладку (с аргументом или без). Данный ключ подобен `-d`. Примеры: *sh(1)*, *uucp(1)*.

Извлечь (extract) (с аргументом). В качестве аргумента указываются файлы, которые необходимо извлечь из архива или рабочего комплекта. Примеры: *tar(1)*, *zip(1)*.

-y

Да (yes) (без аргументов). Санкционирует потенциально разрушительные действия, для которых программа обычно запрашивает подтверждение. Примеры: *fsck(1)*, *rz(1)*.

-z

Разрешает сжатие (без аргументов). Часто данный ключ используется в программах архивирования и резервного копирования. Примеры: *bzip(1)*, GNU *tar(1)*, *zcat(1)*, *zip(1)*, *cvs(1)*.

Приведенные выше примеры взяты из инструментария Linux, но соответствуют большинству современных Unix-систем.

Выбирая буквы для параметров командной строки разрабатываемой программы, следует обратиться к справочным руководствам по подобным инструментам и попытаться использовать такие же буквы, какие используются в них для аналогичных функций разрабатываемой программы. Следует заметить, что некоторые отдельные прикладные области имеют особенно жесткие соглашения о ключах командной строки, нарушать которые опасно. Такие соглашения характерны для компиляторов, почтовых агентов, текстовых фильтров, сетевых утилит и программного обеспечения для X Window. Например, разработчик, который написал почтовый агент, использующий ключ `-s` для какой-либо другой функции вместо подстановки темы, услышит справедливую критику в свой адрес.

Стандарты программирования проекта GNU[5] рекомендуют традиционные значения для нескольких параметров с двойным дефисом. В проекте также перечислены длинные параметры, которые используются во многих GNU-программах, хотя они и не стандартизированы. Если программа разрабатывается с использованием GNU-стиля, и какой-либо необходимый параметр обладает функцией, подобной одному из перечисленных параметров, то следует всеми способами соблюсти правило наименьшей неожиданности и использовать имеющееся имя.

10.5.2. Переносимость на другие операционные системы

Для применения параметров командной строки необходимо ее присутствие в системе. В семействе MS-DOS командная строка, несомненно, присутствует, хотя в Windows она скрыта GUI-интерфейсом и используется слабо. Тот факт, что символом параметра обычно является "/" вместо "-" — только деталь. Классическая MacOS и другие исключительно GUI-среды не имеют близкого эквивалента параметров командной строки.

---

5 См. стандарты GNU-программирования на странице <http://www.gnu.org/prep/standards.html>.

10.6. Выбор метода

Выше последовательно были рассмотрены системные и пользовательские файлы конфигурации, переменные окружения и аргументы командной строки — направление от конфигурации, которую изменить сложнее, к конфигурации, изменяемой проще. Имеется четкое соглашение о том, что удобные Unix-программы, использующие несколько мест хранения конфигурационных сведений, должны просматривать их в указанном порядке, позволяя более поздним установкам переназначать более ранние (существуют специфические исключения, такие как параметры командной строки, определяющие местонахождение файла профиля).

В частности, настройки окружения обычно имеют превосходство над установками файлов конфигурации, но могут быть переназначены параметрами командной строки. Хорошей практикой является предоставление параметра командной строки, подобного ключу `-e` в утилите *make(1)*, который может переназначить установки окружения или объявления в конфигурационных файлах. Данный способ позволяет включать программу в сценарии со строго определенным поведением независимо от вида файлов конфигурации или значений переменных окружения.

Выбор мест хранения настроек зависит от количества постоянных конфигурационных параметров, которые должны сохраняться между вызовами программы. Программы, разработанные преимущественно для использования в неинтерактивном режиме (как, например, генераторы или фильтры в конвейерах), обычно полностью конфигурируются с помощью параметров командной строки. В число хороших примеров данной модели включаются утилиты *ls(1)*, *grep(1)* и *sort(1)*. Другой крайний случай — крупные программы со сложным диалоговым поведением могут полностью зависеть от файлов конфигурации и переменных окружения, и в обычном использовании требуют только несколько параметров командной строки или не требуют их вообще. Большинство диспетчеров окон системы X представляют собой хорошие примеры такой модели.

(В операционной системе Unix файл может иметь несколько имен или “ссылок”. Во время запуска каждая программа может определить имя, посредством которого она была вызвана. Другой способ указать программе, имеющей несколько режимов работы, в каком из них она должна запуститься, заключается в создании ссылки для каждого режима. Программа обнаруживает ссылку, через которую она была вызвана, и соответствующим образом изменяет режим работы. Однако данная техника обычно считается неаккуратной и используется нечасто.)

Ниже рассматриваются две программы, которые учитывают информацию, собранную из всех трех групп конфигурационных данных. Читателям будет полезно обдумать, почему для каждого блока конфигурационных данных информация накапливается именно таким способом.

10.6.1. Учебный пример: *fetchmail*

Программа *fetchmail* использует только две переменные среды — `USER` и `HOME`. Данные переменные входят в состав предопределенного набора конфигурационных данных, инициализируемых системой, и используются многими программами.

Значение переменной HOME используется для поиска файла профиля .fetchmailrc, который содержит конфигурационную информацию, описанную с помощью довольно сложного синтаксиса, подчиняющегося shell-подобным лексическим правилам, описанным выше. В данном случае такой подход целесообразен, поскольку после первоначальной установки конфигурация fetchmail изменяется исключительно редко.

Не существует ни /etc/fetchmailrc, ни какого-либо другого специфичного для fetchmail общесистемного файла. Обычно в таких файлах содержится конфигурация, которая не является специфичной для отдельного пользователя. В fetchmail действительно используется небольшая группа подобных свойств, в частности, имя локального почтмейстера (postmaster), несколько ключей и значений, описывающих настройку локального почтового транспорта (например, номер порта локального SMTP-слушателя). Однако на практике данные значения редко отклоняются от стандартных значений, настроенных на этапе компиляции. Если они изменяются, то это часто происходит определенным для пользователя способом. Поэтому в общесистемном конфигурационном файле fetchmail нет никакой необходимости.

Программа fetchmail может извлекать тройки узел/ имя/пароль из файла .netrc. Таким образом, программа получает сведения аутентификации наименее неожиданным способом.

В fetchmail имеется развитый набор параметров командной строки, близко, но не полностью дублирующих установки, которые могут быть выражены в файле .fetchmailrc. Данный набор первоначально не был большим, однако со временем он разрастался, по мере того как в мини-язык .fetchmailrc добавлялись новые конструкции, а также более или менее автоматически добавлялись параллельные параметры командной строки.

Цель поддержки всех данных параметров заключалась в том, чтобы сделать программу fetchmail более простой для включения в сценарии, позволяя пользователям переназначать с помощью командной строки настройки, определенные в файлах конфигурации. Но оказалось, что за исключением нескольких параметров, таких как --fetchall и –verbose, требуются очень не многие имеющиеся параметры. Кроме того, в ходе использования программы вообще не возникает потребностей, которые невозможно было бы удовлетворить с помощью сценария, создающего на лету временный конфигурационный файл и передающего его в fetchmail с помощью ключа -f.

Таким образом, большинство параметров командной строки никогда не используются, и их включение, вероятно, было ошибкой. Они несколько загромождают код fetchmail, не давая какого-либо полезного эффекта.

> Если бы загромождение кода было единственной проблемой, то никто, кроме нескольких кураторов, не беспокоился бы. Однако параметры увеличивают вероятность появления ошибок в коде, особенно ввиду непредвиденного взаимодействия между редко используемыми параметрами. Еще хуже то, что они значительно загромождают справочные руководства, которые обременяют всех.
>
> Дуг Макилрой

Из всего вышесказанного можно извлечь следующий урок: если бы я достаточно тщательно продумал модель использования программы fetchmail и меньше придерживался специализированного подхода в добавлении функций, то дополнительной сложности можно было бы избежать.

> Альтернативный способ исправления подобных ситуаций, который не загромождает ни код, ни руководство, заключается в использовании ключа "установить переменную параметра", такого как ключ `-O` в *sendmail*, позволяющий указать имя и значение параметра и устанавливающий значение данного параметра так, как если бы оно было задано в конфигурационном файле. Более мощный вариант данной функции имеется в программе *ssh* с (ключ `-o`) : аргумент ключа `-o` интерпретируется так, как будто он представляет собой строку, добавленную в конец конфигурационного файла. При создании аргумента можно использовать весь доступный конфигурационный синтаксис. Любой из таких подходов предоставляет пользователям с необычными требованиями способ переназначать конфигурацию из командной строки, не требуя при этом от разработчика создания отдельного параметра для каждого конфигурационного значения.
>
> Генри Спенсер.

10.6.2. Учебный пример: сервер XFree86

Система оконного интерфейса X представляет собой ядро, поддерживающее растровые дисплеи на Unix-машинах. Unix-приложения, запущенные на клиентской машине с растровым дисплеем, получают входные события посредством системы X и отправляют ей запросы на создание экранных изображений. Многих ставит в тупик то, что "серверы" X фактически запускаются на клиентской машине — они существуют для обслуживания запросов на взаимодействие с дисплеем клиентской машины. Приложения, отправляющие такие запросы X-серверу, называются "X-клиентами", несмотря на то, что они могут быть запущены на серверной машине. Невозможно без путаницы объяснить эту "перевернутую" терминологию.

X-серверы имеют неприступно сложный интерфейс к своему окружению. Это не удивительно, поскольку им приходится взаимодействовать с широким диапазоном сложного аппаратного обеспечения и пользовательских настроек. Запросы окружения являются общими для всех X-серверов, документированными в справочных руководствах *X(1)* и *Xserver(1)*, а значит, представляют собой полезный пример для изучения. Ниже рассматривается XFree86, реализация системы X, применяемая в операционной системе Linux и других Unix-системах с открытым исходным кодом.

Во время запуска сервер XFree86 изучает общесистемный конфигурационный файл. Точный путь к данному файлу варьируется в различных сборках X на разных платформах, но базовое имя постоянно — XF86Config. Данный файл имеет shell-подобный синтаксис, как было описано выше. Ниже приводится пример одного из разделов файла XF86Config.

Пример 10.2. Конфигурация X

```
# VGA-сервер 16 цветов

Section "Screen"
    Driver      "vga16"
    Device      "Generic VGA"
    Monitor     "LCD Panel 1024x768"
    Subsection  "Display"
        Modes       "640x480" "800x600"
        ViewPort    0 0
    EndSubsection
EndSection
```

В файле XF86Config описывается аппаратное обеспечение машины (графическая плата, монитор), клавиатура и указательное устройство (мышь/трекбол/сенсорная панель). Хранение данных сведений в общесистемном конфигурационном файле целесообразно, поскольку они касаются всех пользователей машины.

Как только X-сервер получил конфигурацию аппаратного обеспечения из конфигурационного файла, он использует значение переменной окружения `HOME` для обнаружения двух файлов профиля, расположенных в начальном каталоге вызывающего пользователя, `.Xdefaults` и `.xinitrc`[6].

В файле `.Xdefaults` указываются специализированные ресурсы данного пользователя, относящиеся к системе X (в число простейших примеров можно включить шрифт и цвета фона и переднего плана для эмулятора терминала). Однако выражение "относящиеся к системе X" указывает на проблему проектирования. Группировать описания всех данных ресурсов в одном месте удобно для проверки и редактирования, но не всегда ясно, какие ресурсы следует объявлять в файле `.Xdefaults`, и какие параметры необходимо хранить в файле профиля приложения. В файле `.xinitrc` определяются команды, которые должны быть выполнены для инициализации пользовательского рабочего стола в X сразу после запуска сервера. В число данных программ почти всегда включается диспетчер окон или диспетчер сеанса.

X-серверы имеют большой набор параметров командной строки. Некоторые из них, такие как `-fp` (font path — путь к шрифтам), имеют преимущество перед параметрами файла XF86Config. Некоторые, например, `-audit`, предназначены для того, чтобы облегчить отслеживание ошибок сервера. Если данные параметры вообще используются, то, скорее всего, очень часто изменяются между тестовыми запусками, а следовательно, являются маловероятными кандидатами на включение в конфигурационный файл. Очень важным является параметр, устанавливающий номер дисплея сервера. На узле могут быть запущены несколько серверов, если каждому из них предоставляется уникальный номер дисплея, но все экземпляры совместно используют один и тот же конфигурационный файл (или файлы), поэтому номер дисплея невозможно получить исключительно из данных файлов.

6 Файл `.xinitrc` является аналогом каталога автозагрузки в Windows и других операционных системах.

10.7. Нарушение правил

Описанные в данной главе соглашения не абсолютны, но их нарушение в будущем усугубить разногласия между пользователями и разработчиками. В случае крайней необходимости их можно нарушить, однако, прежде чем это сделать, необходимо точно знать, для чего это необходимо. Разработчику, нарушающему данные соглашения, следует убедиться, что решение задач традиционными способами невозможно и что он имеет полное представление об ошибках в соответствии с правилом исправности.