ПРАВА ДОСТУПА К ФАЙЛАМ. КАКИЕ ПРАВА НУЖНЫ ДЛЯ РЕДАКТИРОВАНИЯ ФАЙЛА? ДЛЯ ЕГО ЧТЕНИЯ?

Unix каждому файлу соответствует набор прав доступа, представленный в виде 9-ти битов режима. Он определяет, какие пользователи имеют право читать файл, записывать в него данные или выполнять его. Вместе с другими тремя битами, влияющими на запуск исполняемых файлов, этот набор образует *код режима доступа к файлу*. Двенадцать битов режима хранятся в 16-битовом поле индексного дескриптора вместе с 4-мя дополнительными битами, определяющими тип файла. Последние 4 бита устанавливаются при создании файлов и не подлежат изменению. Биты режима (далее права) могут изменяться либо владельцем файла, либо суперпользователем с помощью команды **chmod**.

1) Право на **чтение (r)** файла означает, что пользователь может просматривать содержимое файла с помощью различных команд просмотра, например, командой more или с помощью любого текстового редактора. Но, подредактировав содержимое файла в текстовом редакторе, вы не сможете сохранить изменения в файле на диске, если не имеете права на **запись (w)** в этот файл. Право на **выполнение (x)** означает, что вы можете загрузить файл в память и попытаться запустить его на выполнение как исполняемую программу. Конечно, если в действительности файл не является программой (или скриптом shell), то запустить этот файл на выполнение не удастся, но, с другой стороны, даже если файл действительно является программой, но право на выполнение для него не установлено, то он тоже не запустится.

2) Естественно, что по отношению к каталогам трактовка понятий "право на чтение", "право на запись" и "право на выполнение" несколько изменяется. Право на чтение по отношению к каталогам легко понять, если вспомнить, что каталог — это просто файл, содержащий список файлов в данном каталоге. Следовательно, если вы имеете право на **чтение каталога**, то вы можете просматривать его содержимое (этот самый список файлов в каталоге). **Право на запись** тоже понятно — имея такое право, вы сможете создавать и удалять файлы в этом каталоге, т. е. просто добавлять в каталог или удалять из него запись, содержащую имя какого-то файла и соответствующие ссылки. Право на выполнение интуитивно менее понятно. Оно в данном случае означает право переходить в этот каталог. Если вы, как владелец, хотите дать доступ другим

пользователям на просмотр какого-то файла в своем каталоге, вы должны дать им право доступа в каталог, т. е. дать им "**право на выполнение каталога**". Более того, надо дать пользователю право на выполнение для всех каталогов, стоящих в дереве выше данного каталога. Поэтому в принципе для всех каталогов по умолчанию устанавливается право на выполнение как для владельца и группы, так и для всех остальных пользователей. И, уж если вы хотите закрыть доступ в каталог, то лишите всех пользователей (включая группу) права входить в этот каталог. Только не лишайте и себя такого права, а то придется обращаться к суперпользователю!

3) Чтобы прочитать файл из каталога, зная путь до него, необходимо право для перехода в директорию (x) и права на чтение файла. (права на чтение директории не обязательны)

Стоит выучить таблицу

воичная	**восьмеричная**	**символьная**	**права на файл**	**права на каталог**
000	0	---	нет	нет
001	1	--x	выполнение	чтение файлов и их свойств
010	2	-w-	запись	нет
011	3	-wx	запись и выполнение	всё, кроме чтения списка файлов
100	4	r--	чтение	чтение имён файлов
101	5	r-x	чтение и выполнение	доступ на чтение
110	6	rw-	чтение и запись	чтение имён файлов
111	7	rwx	все права	все права

ДОПОЛНИТЕЛЬНЫЕ ПРАВА

Существуют также специальные биты, такие как **SUID**, **SGID** и **Sticky**-бит. **SUID**, **SGID** влияют на запуск файла (только от имени владельца и группы соответственно), а **Sticky** влияет на определение владельца объектов в каталоге (удаление только владельцем). При их применении необходимо использовать не три восьмеричных цифры, а 4. Зачастую, в различной технической литературе права обозначаются именно 4-мя цифрами, например **0744**.

- **SUID – 4000** и л и **u+s** (rw**s**------)
- **SGID – 2000** и л и **g+s** (---rw**s**---)
- **sticky- б и т – 1000** и л и **+t** Символ «**t**» может быть как строчная буква (**t**), так и прописная (**T**). Строчная буква отображается в том случае, если перед установкой **sticky bit** произвольный пользователь уже имел право на выполнение (**x**), а прописная (T) — если такого права у него не было. Конечный результат один и тот же, но регистр символа дает дополнительную информацию об исходных установках.

ПОЛЬЗОВАТЕЛИ И ГРУППЫ

В основе механизмов разграничения доступа лежат имена пользователей и имена групп пользователей. Вы уже знаете, что в Linux каждый пользователь имеет уникальное имя, под которым он входит в систему (логируется). Кроме того, в системе создается некоторое число групп пользователей, причем каждый пользователь может быть включен в одну или несколько групп. Создает и удаляет группы суперпользователь, он же может изменять состав участников той или иной группы. Члены разных групп могут иметь разные права по доступу к файлам, например, группа администраторов может иметь больше прав, чем группа программистов.

Группы были разработаны для того, чтобы расширить возможности управления правами.

Все группы, созданные в системе, находятся в файле /etc/group. Посмотрев содержимое этого файла, вы можете узнать список групп linux, которые уже есть в вашей системе. И вы будете удивлены.

Пользователь имеет основную группу, она указывается при создании, а также несколько дополнительных. Основная группа отличается от обычных тем, что все файлы в домашнем каталоге пользователя имеют эту группу, и при ее смене, группа этих каталогов тоже поменяется. Также именно эту группу получают все файлы созданные пользователем. Дополнительные группы

нужны, чтобы мы могли разрешить пользователям доступ к разным ресурсам добавив его в эти группы в linux.
Управление группами Linux для пользователя выполняется с помощью команды usermod. Рассмотрим ее синтаксис и опции:
$ usermod опции имя_пользователя

- **-G** — дополнительные группы, в которые нужно добавить пользователя
- **-g** изменить основную группу для пользователя
- **-R** удалить пользователя из группы.

При вызове функции ls –l указывается основная группа пользователя
+В конце списка прав указывает на наличие установленных права ACL (access control list - списки контроля доступа) - списки контроля доступа

ПОТОКИ ВВОДА-ВЫВОДА И ИХ ПЕРЕНАПРАВЛЕНИЕ

По принятым соглашениям все командные оболочки при запуске новой программы открывают для нее три файловых дескриптора: файл стандартного ввода, файл стандартного вывода и файл стандартного вывода сообщений об ошибках. За исключением особых случаев, все три дескриптора по умолчанию связаны с терминалом,
Потоками называются файлы (всё есть файл!), с которыми можно обмениваться информацией.

Стандартные потоки и их назначение:

- STDIN - ввод информации
- STDOUT - вывод информации
- STDERR - вывод ошибок

ТЕРМИНАЛ – ИНТЕРАКТИВНАЯ КОМАНДНАЯ ОБОЛОЧКА

Для обеспечения интерфейса командной строки в операционных системах часто используются командные интерпретаторы, которые могут представлять собой самостоятельные языки программирования, с собственным синтаксисом и отличительными функциональными возможностями

Командная оболочка — это отдельный программный продукт, который обеспечивает прямую связь между пользователем и операционной системой. Текстовый пользовательский интерфейс командной строки предоставляет среду, в которой выполняются приложения и служебные программы с текстовым интерфейсом. В командной оболочке программы выполняются, и результат выполнения отображается на экране.

Интерпретатор командной строки, который считывает ввод пользователя и выполняет команды. Ввод осуществляется посредством терминала или считывается из файла (называется сценарием командной оболочки)

В совокупности с набором утилит, оболочка представляет собой операционную среду, язык программирования и средство решения как системных, так и некоторых прикладных задач, в особенности, автоматизации часто выполняемых последовательностей команд.

Интерпретаторы — трансляторы языков программирования, работают на отличающемся от компиляторов принципе. Интерпретаторы не производят исполняемого машинного кода. Они берут исходный текст программы на языке программирования и выполняют его сами строка за строкой. При этом интерпретатор извлекает из файла с исходным текстом одну команду, распознает ее и вызывает те или иные функции операционной системы. Интерпретатор определяет команду и переводит (интерпретирует) ее так,

чтобы операционная система поняла, что от нее хотят. Скорость выполнения программ в режиме интерпретации намного ниже, чем у компилированного кода, за счет того, что работа программы идет не напрямую с центральным процессором на языке машинных команд, а через программу-посредника, которая и тратит большое количество времени на распознавание исходного текста. В отличие от интерпретаторов, компиляторы «знакомятся» с исходными текстами программы всего один раз, когда делают из текста на языке программирования машинный код.
Простые интерпретаторы анализируют и выполняют (интерпретируют) программу последовательно (покомандно или построчно). Синтаксические ошибки обнаруживаются, когда интерпретатор приступает к выполнению команды (строки) содержащей ошибку. Сложные интерпретаторы компилирующего типа перед выполнением производят компиляцию исходного кода программы в машинный или «промежуточный код». Они быстрее выполняют большие и циклические программы, не занимаются анализом исходного кода в реальном времени. Некоторые интерпретаторы для начинающих программистов (преимущественно, для языка Бейсик) могут работать в режиме диалога, добавляя вводимую строку команд в программу (в памяти) или выполняя команды непосредственно.

Интерфейс командной строки (англ. Command line interface, CLI) — разновидность текстового интерфейса (CUI) между человеком и компьютером, в котором инструкции компьютеру даются в основном путём ввода с клавиатуры текстовых строк (команд), в UNIX-системах возможно применение мыши. Также известен под названием **консоль**.

В UNIX-подобных системах наиболее распространены такие языки командных интерпретаторов как bash, sh и ksh, но также применяются альтернативные оболочки zsh, csh и tcsh, отличающиеся синтаксисом управляющих конструкций и поведением переменных.

ФАЙЛОВАЯ СИСТЕМА

Файловая система UNIX представляет собой иерархическую древовидную структуру, состоящую из каталогов и файлов. Начинается с каталога, называемого корнем “/”

Каталог – файл, в котором содержатся каталожные записи. Каждая запись – структура из имени файла и доп информации, описывающей атрибуты файла:

- размер файла в байтах;
- идентификатор устройства;
- идентификатор владельца файла;
- идентификатор группы-владельца файла;
- режим доступа к файлу, определяющий кто и какой доступ имеет к файлу;
- дополнительные системные и пользовательские флаги, которые дополнительно могут ограничивать доступ к файлу и его модификацию;
- таймштампы, отражающие время модификации индексного дескриптора (*ctime, changing time*), время модификации содержимого файла (*mtime, modification time*) и время последнего доступа к файлу (*atime, access time*);
- счётчик для учёта количества жёстких ссылок на файл;
- указатели на физические блоки диска, в которых хранится содержимое файла (об этом ниже).

Файловая система, как правило, состоит из двух частей:

1. Метаданные (данные о данных).
2. Сами данные.

ИМЕНА ФАЙЛОВ

Имена элементов каталога – **имена файлов.** Только два символа не могут встречаться в имени файла – прямой слэш (/) и нулевой символ (\0). Символ слэша разделяет имена файлов, из которых состоит строка пути к файлу, а нулевой описывает конец строки. Специальными символами командного интерпретатора называть файлы не рекомендуется, при использовании необходимо применять экранирование (\ или одинарные кавычки)

Полное*,* или **абсолютное***,* имя файла рекурсивно определяется как полное имя каталога, в котором он содержится, за которым следует слеш и имя файла.

Длина имени файла в современных ФС может быть длиной до 255 символов.

РАБОЧИЙ КАТАЛОГ

У каждого процесса имеется **рабочий каталог** (текущий рабочий каталог) – Каталог, от которого отсчитываются все относительные пути. Процесс может менять его при помощи функции **chdir (cd). Относительное имя файла** определяет его путь из текущего, а не корневого каталога, и не начинается со слеша.

При входе пользователя в систему рабочий каталог – его домашний каталог. Рабочий каталог содержится в переменной окружения $PWD.

ИНДЕКСНЫЕ ДЕСКРИПТОРЫ

Или inode. Метаданные являются ключевыми в организации файловых систем. Они содержат информацию о данных на диске - атрибуты. Роль метаданных крайне важна, поскольку без них файловая система представляла бы из себя лишь набор байт, в котором невозможно было бы определить что и где физически находится на диске.

В общем случае в файловых системах операционных систем *NIX с каждым файлом и каталогом связан соответствующий дескриптор — **inode**, который обычно обозначается целым числом и в котором хранятся метаданные.

Некоторые файловые системы создают дескрипторы в момент создания файловой системы, в результате располагая фиксированным количеством индексных дескрипторов, то есть фиксированным пределом количества файлов, хранящихся в ФС. Так, например, работает файловая система *Ext-3*. Таким образом получается, что вы в какой-то момент не сможете создать файл, даже если свободного пространства на диске будет достаточно. Такое

случается крайне редко, но, тем не менее, не исключено. Если, используя такую файловую систему, вам понадобиться больше индексных дескрипторов, вам придётся заново создавать ФС, средств увеличить количество inode без потери данных нет.

Индексные дескрипторы не являются чем-то мистическим, они являются частью Linux. Вы можете увидеть их присутствие, например при помощи команды *'ls -i'*:

ТИПЫ ФАЙЛОВ

При вызове функции ls –l первый символ указывает на тип файла

- - = — обычный файл;
- d = — каталог;
- b = — файл блочного устройства;
- c = — файл символьного устройства;
- s = — доменное гнездо (socket);
- p = — именованный канал (pipe);
- l = — символическая ссылка (link).

Обычные файлы (-)

Сюда относятся все файлы с данными, играющими роль ценной информации сами по себе. Linux все-равно текстовый перед ней файл или бинарный. В любом случае это будет обычный файл.

Каталоги (d)

Это файлы, в качестве данных которых выступают списки других файлов и каталогов. Именно в данных каталога осуществляется связь имени файла (словесного обозначения для людей) с его индексным дескриптором (истинным именем-числом). Отсюда следует, что один и тот же файл может существовать под разными именами и/или в разных каталогах: все имена будут связаны с одним и тем же индексным дескриптором (механизм жестких ссылок). Также следует, что файлы всегда содержаться в каталогах, иначе просто недоступны.

Символьная ссылка (l) — это файл в данных которого, содержится указание на адрес другого файла по его имени (но не индексному дескриптору). См. в разделе «ССЫЛКИ»

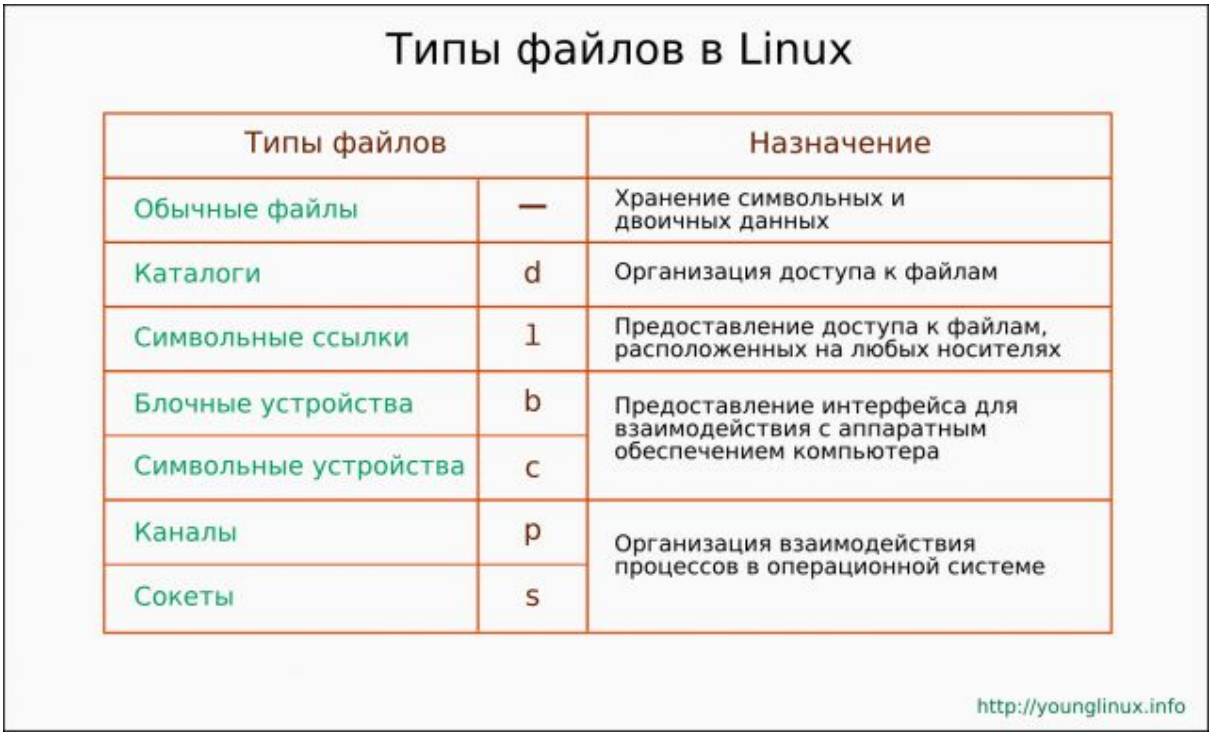
Символьные (c) и блочные устройства (b)

Файлы устройств предназначены для обращения к аппаратному обеспечению компьютера (дискам, принтерам, терминалам и др.). Когда происходит обращение к файлу устройства, то ядро операционной системы передает запрос драйверу этого устройства.

К символьным устройствам обращение происходит последовательно (символ за символом). Примером символьного устройства может служить терминал. Считывать и записывать информацию на блочные устройства можно в произвольном порядке, причем блоками определенного размера. Пример: жесткий диск.

Сокеты (s) и каналы (p)

Для того, чтобы понять что такое канал и сокет и для чего они нужны, необходимо понимание что такое процесс в операционной системе. И каналы и сокеты организуют взаимодействие процессов. Пользователь с данными типами файлов почти никогда не сталкивается.



ССЫЛКИ

С помощью ссылок п *NIX системах возможно размещать один и тот-же файл в разных директориях. В Linux существует два типа ссылок на файлы. Это символические и жесткие ссылки Linux.

СИМВОЛИЧЕСКИЕ ССЫЛКИ

Символические ссылки более всего похожи на обычные **ярлыки**. Они содержат адрес нужного файла в вашей файловой системе. Когда вы пытаетесь открыть такую ссылку, то открывается целевой файл или папка. Главное ее отличие от жестких ссылок в том, что при удалении целевого файла ссылка останется, но она будет указывать в никуда, поскольку файла на самом деле больше нет.
Вот основные особенности символических ссылок:

- Могут ссылаться на файлы и каталоги;
- После удаления, перемещения или переименования файла становятся недействительными;
- **ПРАВА ДОСТУПА И НОМЕР INODE ОТЛИЧАЮТСЯ ОТ ИСХОДНОГО ФАЙЛА;**
- При изменении прав доступа для исходного файла, права на ссылку останутся неизменными;
- Можно ссылаться на другие разделы диска;
- Содержат только имя файла, а не его содержимое.

ЖЕСТКИЕ ССЫЛКИ

Этот тип ссылок реализован на более низком уровне файловой системы. Файл размещен только в определенном месте жесткого диска. Но на это место могут ссылаться несколько ссылок из файловой системы. Каждая из ссылок — это отдельный файл, но ведут они к одному участку жесткого диска. Файл можно перемещать между каталогами, и все ссылки останутся рабочими, поскольку для них неважно имя. Рассмотрим особенности:

- Работают только в пределах одной файловой системы;
- Нельзя ссылаться на каталоги;
- **ИМЕЮТ ТУ ЖЕ ИНФОРМАЦИЮ INODE И НАБОР РАЗРЕШЕНИЙ ЧТО И У ИСХОДНОГО ФАЙЛА;**
- Разрешения на ссылку изменяться при изменении разрешений файла;
- Можно перемещать и переименовывать и даже удалять файл без вреда ссылке.
- Если для одного из файлов поменять разрешения, то они изменяться и у другого. Теперь удалите исходный файл:

Возможность существования нескольких прямых ссылок на каталог могла бы приводить к утечкам памяти на диске. На каталоги можно делать символьные ссылки.
Однако при создании директории мы автоматически создаём жесткие ссылки на саму директорию «.» и на родительскую директорию «..». Для корневой директории «..» ссылается на саму себя.

Если файл имеет несколько жестких ссылок, то он удаляется только тогда, когда удаляется последняя ссылка, указывающая на его inode, и счетчик ссылок сбрасывается до 0.

$ ln опции файл_источник файл_ссылки

Рассмотрим опции утилиты:

-d — разрешить создавать жесткие ссылки для директорий суперпользователю;
-f — удалять существующие ссылки;
-i — спрашивать нужно ли удалять существующие ссылки;
-P — создать жесткую ссылку;
-r — создать символическую ссылку с относительным путем к файлу;
-s — создать символическую ссылку.

Нельзя создавать жесткие ссылки на директории по требованиям POSIX (набор стандартов, разработанных комитетом ISO)

Считается выходом за предел файловой системы

ОКРУЖЕНИЕ

Каждый запускаемый процесс система снабжает неким информационным пространством, которое этот процесс вправе изменять как ему заблагорассудится. Правила пользования этим пространством просты: в нём можно задавать именованные хранилища данных (переменные окружения), в которые записывать какую угодно информацию (присваивать значение переменной окружения), а впоследствии эту информацию считывать (подставлять значение переменной). Дочерний процесс — точная копия родительского, поэтому его окружение — также точная копия родительского. Если про дочерний процесс известно, что он использует значения некоторых переменных из числа передаваемых ему с окружением, родительский может заранее указать, каким из копируемых в окружении переменных нужно изменить значение. При этом, с одной стороны, никто (кроме системы, конечно) не сможет вмешаться в процесс передачи данных, а с другой стороны, одна и та же утилита может быть использована одним и тем же способом, но в изменённом окружении — и выдавать различные результаты

Каждый процесс имеет параметры и окружение. Параметры процесса — это массив строк, кончающийся нулевым указателем, называемый обычно argv и передаваемый в первую вызванную функцию. Количество параметров называется argc. Окружение процесса — это массив строк типа ПЕРЕМЕННАЯ=значение, тоже кончающийся нулевым указателем и доступный

через глобальную переменную environ, имеющую (в C) тип char** . Посмотреть окружение можно командой printenv [имя] (если имя не указано, выводится всё окружение), а в командной оболочке sh или bash - ещё и встроенной командой

Переменные окружения — именованные переменные, содержащие текстовую информацию, которую могут использовать запускаемые программы. Такие переменные могут содержать общие настройки системы, параметры графической или командной оболочки, данные о предпочтениях пользователя и многое другое. Значением такой переменной может быть, например, место размещения исполняемых файлов в системе, имя предпочитаемого текстового редактора или настройки системной локали. Пакет coreutils содержит программы printenv и env. Чтобы отобразить список текущих переменных окружения, используйте printenv, которая отобразит имена и значения каждой переменной окружения:

Некоторые важные переменные

PATH содержит список каталогов, в которых система ищет исполняемые файлы. Когда обычная команда, например, ls, rc-update или emerge, интерпретируется командной оболочкой (такой как bash или zsh), оболочка ищет исполняемый файл с указанным именем в этом списке, и, если находит, запускает файл, передав ему указанные аргументы командной строки. Чтобы запускать исполняемые файлы, пути к которым не находятся в PATH, необходимо указывать полный путь к файлу, например /bin/**ls.**

HOME содержит путь к домашнему каталогу текущего пользователя. Эта переменная может использоваться приложениями для определения расположения файлов настроек пользователя, который их запускает.

PWD содержит путь к рабочему каталогу.

OLDPWD содержит путь к предыдущему рабочему каталогу, то есть, значение PWD перед последним вызовом cd.

SHELL содержит имя текущей командной оболочки, например, bash.

TERM содержит имя запущенной программы-терминала, например xterm.

PAGER указывает команду для запуска программы постраничного просмотра содержимого текстовых файлов, например, /bin/less.

EDITOR содержит команду для запуска программы для редактирования текстовых файлов, например /usr/bin/nano. Также можно задать специальную команду, которая будет выбирать редактор в зависимости от окружения, например, gedit в X или nano терминале.

MANPATH содержит список каталогов, которые использует man для поиска man-страниц. Стандартным значением является /usr/share/man:/usr/local/share/man

TZ может использоваться для установки временной зоны. Доступные временные зоны можно найти в /usr/share/zoneinfo/

Установка переменных

На системном уровне

Большинство дистрибутивов Linux советуют изменять или добавлять переменные окружения в /etc/profile или других местах. Имейте в виду, что сразу множество файлов могут содержать переменные окружения и переопределять их. По сути, любой скрипт может быть использован для этого, однако, по принятым в UNIX соглашениям, следует использовать для этого только определенные файлы.

Следующие файлы следует использовать для установки переменных окружения на уровне системы: /etc/profile, /etc/bash.bashrc и /etc/environment. Каждый из этих файлов имеет свои ограничения, поэтому следует внимательно выбирать тот, который подходит для ваших целей.

/etc/profile устанавливает переменные только для командных оболочек. Он может запускать любые скрипты в оболочках, совместимых с Bourne shell.
/etc/bash.bashrc устанавливает переменные только для интерактивных оболочек. Он также запускает bash-скрипты.
/etc/environment используется модулем PAM-env. Здесь можно указывать только пары имя=значение.

На уровне пользователя

Вам не всегда нужно будет устанавливать переменные окружения на уровне системы. Например, вы можете добавить ваш каталог /home/пользователь/bin в PATH, однако, не хотите, чтобы это затрагивало других пользователей системы. Переменные окружения пользователя можно устанавливать во многих других файлах:

Файлы инициализации командной оболочки, например
~/.profile используется также многими оболочками,

~/.pam_environment пользовательский аналог файла /etc/environment, который используется модулем PAM-env. Смотрите подробнее в pam_env(8).
Например, чтобы добавить каталог в PATH, поместите следующее в **~/.bash_profile**

АРГУМЕНТЫ КОМАНДНОЙ СТРОКИ

Одной из важнейших возможностей командной оболочки является возможность обработки строк команд. При вводе команды после приглашения командной оболочки и нажатии клавиши Enter командная оболочка приступает к обработке строки команды, разделяя ее на аргументы. При обработке строки команды командная оболочка может внести множество изменений в переданные вами аргументы.

Данный процесс называется раскрытием команд командной оболочки. После того, как командная оболочка заканчивает обработку и модификацию переданной строки команды, будет осуществляться непосредственное исполнение результирующей команды.

Бывает, что данные в программу передаются из командной строки при ее вызове. Такие данные называются аргументами командной строки.

Если программа написана на языке C, то при ее запуске управление сразу передается в функцию main(), следовательно, именно она получает аргументы командной строки, которые присваиваются ее переменным-параметрам.

При вызове программы из командной строки в нее всегда передается пара данных:

- целое число, обозначающее количество слов (элементов, разделенных пробелами) в командной строке при вызове
- указатель на массив строк, где каждая строка — это отдельное слово из командной строки.

Обратите внимание на терминологию, есть всего два аргумента программы (число и массив), но сколько угодно аргументов командной строки. Аргументы командной строки "преобразуются" в аргументы программы (в аргументы функции main()).

Эти данные (число и указатель) передаются в программу даже тогда, когда она просто вызывается по имени без передачи в нее чего-либо

То, что в программу передаются данные, вовсе не означает, что функция main() должна их принимать. Если функция main() определена без параметров, то получить доступ к аргументам командной строки невозможно. Хотя ничего вам не мешает их передавать. Ошибки не возникнет.

Чтобы получить доступ к переданным в программу данным, их необходимо присвоить переменным. Поскольку аргументы сразу передаются в main(), то ее заголовок должен выглядеть таким образом:
main (int n, char *arr[])

В первой переменной (n) содержится количество слов, а во второй — указатель на массив строк. Часто второй параметр записывают в виде **arr. Однако это то же самое. Вспомним, что сам массив строк, содержит в качестве своих элементов указатели на строки. А в функцию мы передаем указатель на первый элемент массива. Получается, что передаем указатель на указатель, т.е. **arr.

Просто дурацкие вопросы:

1) Как создать файл, содержащий в названии пробел? Необходимо взять название файла в кавычки, т.е. :
   - touch ‘a b’
   - touch “a b”
2) Как создать файл, содержащий в названии перенос строки? Необходимо использовать кавычки (одинарные, либо двойные
   - touch ‘a
     b’
   - touch ‘a

     b’
3) Как создать несколько директорий сразу?
   - mkdir -p dir1/../dir2/../dir3/
   - mkdir dir1 dir2 dir3
   - mkdir -p dir1/dir2/{dir3,dir4,dir5}
   - mkdir –p dir1/dir2 dir1/dir3 dir1/dir4
4) Что быстрее: копирование или перемещение?

ср копирование работает медленнее, потому что пробегается по всем копируемым файлам и записывают в новые inode, а mv просто переписывает пути

5) Какое максимальное количество файлов может быть в каталоге?

Файловая система ext4

- Максимальное количество файлов: $2^{32} - 1$ (4 294 967 295 = int)
- Максимальное количество файлов в каталоге: неограниченное

Файловая система ext3

- Максимальное количество файлов: min (volumeSize / 2^{13} numberOfBlocks)

Файловая система ext2

- Максимальное количество файлов: 10^{18} (18 = степень)
- Максимальное количество файлов в каталоге: ~ 1.3 & times; 10^{20} (проблемы с производительностью до 10000)

6) Что такое тильда?
7) Как рассчитывается размер символьной ссылки?
8) Что такое программы-фильтры?

Фильтры — это команды (или программы), которые воспринимают входной поток данных, производят над ним некоторые преобразования и выдают результат на стандартный вывод (откуда его можно перенаправить куда-то еще по желанию пользователя). К числу команд-фильтров относятся уже упоминавшиеся выше команды `cat`, `more`, `less`, `wc`, `cmp`, `diff`, а также следующие команды.

9) Где и как хранятся аргументы программы и как передаются?

10) Любой вопрос про vi
11) Какие команды и символы обрабатываются shell, а какие нет? Как это определяется?
12) Сигналы
13) Переменные окружения
14) Откуда терминал знает, что нужно запустить файл, путь к которому не указан
15) Если существует программа и файл с одинаковыми именами, что запустится при вызове этого имени? (приоритет исполнения)
16) При каком условии программа называется freedor?
17) ^C ^D. Нажатие ^D передает символ конца файла “EOF”. Имеет смысл при вводе, потому что означает конец ввода. То есть всё введённые до него символы передаются программе и ввод завершается. При нажатии просто в

терминале, он закрывается, именно потому что получает символ прекращения ввода. ^C же передаёт программе сигнал завершения SIGINT.

18) Зачем нужны кавычки?

19) Повтор последней команды !!

20) touch — команда Unix, предназначенная для установки времени последнего изменения файла или доступа в текущее время. Также используется для создания пустых файлов.

21) чем обрабатывается маска * ?

22) как отреагирует программа-фильтр на файл, в названии которого содержится перенос строки

23) man поису по кейвордам, главы, уметь пользоваться

24) как строки хранятся в памяти