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)

Если в лабораторной у вас не хватает прав на какое-то действие по заданию, то необходимо добавить себе только нужные права, а затем вернуть их обратно.

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

CHMOD

(Change mod) - программа измерения прав доступа к файлам.
Синтаксис

chmod опции права /путь/к/файлу

Синтаксис настройки прав такой:

группа_пользователейдействиевид_прав

В качестве действий могут использоваться знаки "+" - включить или "-" - отключить. Рассмотрим несколько примеров:

u+x - разрешить выполнение для владельца;
ugo+x - разрешить выполнение для всех;
ug+w - разрешить запись для владельца и группы;
o-x - запретить выполнение для остальных пользователей;
ugo+rwx - разрешить все для всех;
Но права можно записывать не только таким способом. Есть еще восьмеричный формат записи, он более сложен для понимания, но пишется короче и проще. Я не буду рассказывать как считать эти цифры, просто запомните какая цифра за что отвечает, так проще:

0 - никаких прав;
1 - только выполнение;
2 - только запись;
3 - выполнение и запись;
4 - только чтение;
5 - чтение и выполнение;
6 - чтение и запись;
7 - чтение запись и выполнение.

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** или **u+s** (rw**s**------)
- **SGID – 2000** или **g+s** (---rw**s**---)
- **sticky-бит – 1000** или **+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.

Стандартный ввод при работе пользователя в терминале передается через клавиатуру.
Стандартный вывод и стандартная ошибка отображаются на дисплее терминала пользователя в виде текста.
Ввод и вывод распределяется между тремя стандартными потоками:

- stdin - стандартный ввод (клавиатура)
- stdout - стандартный вывод (экран)
- stderr - стандартная ошибка (вывод ошибок на экран)

Потоки также пронумерованы:

- stdin - 0
- stdout - 1
- stderr - 2

Из стандартного ввода команда может только считывать данные, а два других потока могут использоваться только для записи. Данные выводятся на экран и считываются с клавиатуры, так как стандартные потоки по умолчанию ассоциированы с терминалом пользователя. Потоки можно подключать к чему угодно: к файлам, программам и даже устройствам. В командном интерпретаторе bash такая операция называется перенаправлением:

- < file - Использовать файл как источник данных для стандартного потока ввода
- > file - Направить стандартный поток вывода в файл. Если файл не существует, он будет создан, если существует — перезаписан сверху
- 2> file - Направить стандартный поток ошибок в файл. Если файл не существует, он будет создан, если существует — перезаписан сверху
- >>file - Направить стандартный поток вывода в файл. Если файл не существует, он будет создан, если существует — данные будут дописаны к нему в конец
- 2>>file - Направить стандартный поток ошибок в файл. Если файл не существует, он будет создан, если существует — данные будут дописаны к нему в конец
- &>file или >&file - Направить стандартный поток вывода и стандартный поток ошибок в файл. Другая форма записи: >file 2>&1

- ```
  cat<<'EOF'

  Здесь помещается произвольный текст,
  ```
  ```

в том числе – включающий в себя специальные символы

EOF

Перенаправление до достижения EOF

(По сути любой строки, которую укажем)

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)

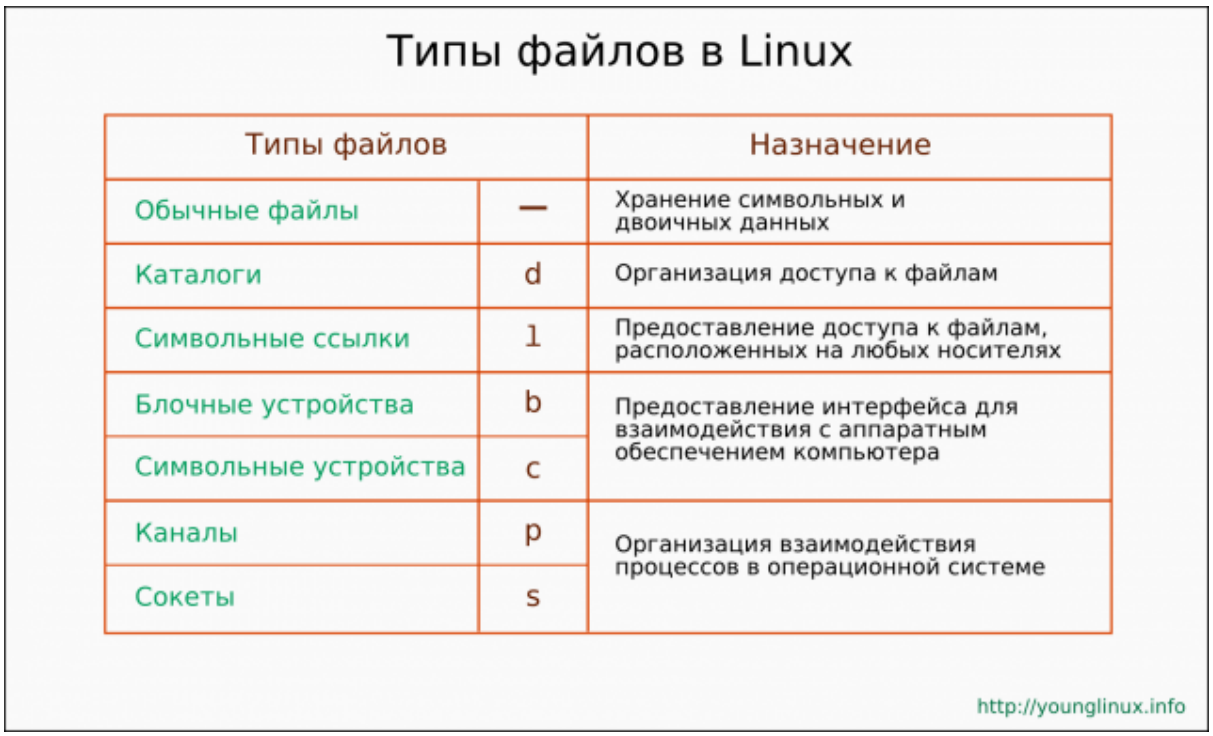
Файлы устройств предназначены для обращения к аппаратному обеспечению компьютера (дискам, принтерам, терминалам и др.). Когда происходит обращение к файлу устройства, то ядро операционной системы передает запрос драйверу этого устройства.

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 в *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)

При создании жесткой ссылки на директорию возникнет неоднозначность при переходе в родительскую папку, так как их будет несколько, плюс может зациклиться рекурсивный поиск, поэтому создание жестких ссылок запрещено.

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.

Если команда set используется с ключом "--", после которого следует переменная, то значение переменной переносится в позиционные параметры (аргументы). Если имя переменной отсутствует, то эта команда приводит к сбросу позиционных параметров.

О команде export http://rus-linux.net/MyLDP/consol/export.html

СИГНАЛЫ

В современных операционных системах существует понятие межпроцессного взаимодействия (Inter-Process Communication – IPC) – это набор способов

обмена данными между процессами и/или потоками. Одним из таких способов обмена служат сигналы.

Сигналы способны в случайное время (асинхронно) прерывать процесс для обработки какого-либо события. Процесс может быть прерван сигналом по инициативе другого процесса или ядра. Ядро использует сигналы для извещения процессов о различных событиях, например о завершении дочернего процесса.

Есть две реализации сигналов - одна сразу прерывает процесс при появлении сигнала и уходит в его обработку, другая выставляет флаг и затем передает управление обработчику, чтобы не возникало проблем с появлением сигнала внутри обработки сигнала.

SIGINT (номер 2) обычно посылается процессу, если пользователь терминала дал команду прервать процесс (обычно эта команда – сочетание клавиш **Ctrl-C**)

SIGABRT (номер 6) посылается программе в результате вызова функции abort(3). В результате программа завершается с сохранением на диске образа памяти.

SIGKILL (номер 9) завершает работу программы. Программа не может ни обработать, ни игнорировать этот сигнал.

SIGSEGV (номер 11) посылается процессу, который пытается обратиться к не принадлежащей ему области памяти. Если обработчик сигнала не установлен, программа завершается с сохранением на диске образа памяти.

SIGTERM (номер 15) вызывает «вежливое» завершение программы. Получив этот сигнал, программа может выполнить необходимые перед завершением операции (например, высвободить занятые ресурсы). Получение SIGTERM свидетельствует не об ошибке в программе, а о желании ОС или пользователя завершить ее.

SIGCHLD (номер 17) посылается процессу в том случае, если его дочерний процесс завершился или был приостановлен. Родительский процесс также получит этот сигнал, если он установил режим отслеживания сигналов дочернего процесса и дочерний процесс получил какой-либо сигнал. По умолчанию сигнал SIGCHLD игнорируется.

SIGCONT (номер 18) возобновляет выполнение процесса, остановленного сигналом SIGSTOP.

SIGSTOP (номер 19) приостанавливает выполнение процесса. Как и SIGKILL, этот сигнал не возможно перехватить или игнорировать.

SIGTSTP (номер 20) приостанавливает процесс по команде пользователя (обычно эта команда – сочетание клавиш Ctrl-Z).

SIGIO/SIGPOLL (в Linux обе константы обозначают один сигнал – номер 29) сообщает процессу, что на одном из дескрипторов, открытых асинхронно, появились данные. По умолчанию этот сигнал, как ни странно, завершает работу программы.

Подробнее о системных сигналах :
https://www.ibm.com/developerworks/ru/library/l-signals_1/index.html

ВАЖНЫЕ СОЧЕТАНИЯ КЛАВИШ

Нажатие **Ctrl + C** заставляет терминал послать сигнал SIGINT процессу, который на данный момент его контролирует. Когда foreground-программа получает сигнал SIGINT, она обязана прервать свою работу.

Нажатие **Ctrl + D** говорит терминалу, что надо зарегистрировать так называемый EOF (end of file – конец файла), то есть поток ввода окончен. Bash интерпретирует это как желание выйти из программы.

Комбинация клавиш **Ctrl + Z** посылает процессу сигнал, который приказывает ему остановиться. Это значит, что процесс остается в системе, но как бы замораживается. Само собой разумеется он уходит в бэкграунд (background) – в фоновый режим. С помощью команды bg его можно снова запустить, оставив при этом в фоновом режиме. Команда fg не только возобновляет ранее приостановленный процесс, но и выводит его из фона на передний план.

ПОДРОБНЕЕ О КОМАНДАХ

В этом разделе будет информация о различных командах UNIX систем.

MAN

ОПЦИИ
-C файл_конфигурации
Указать файл конфигурации для использования; по умолчанию это /etc/man.conf. (См. man.conf(5).)
-M путь
Определить список каталогов для поиска страниц руководства. Каталоги разделяются двоеточиями. Пустой список каталогов равнозначен

неупотреблению -M вовсе. Смотрите ПУТИ ПОИСКА СТРАНИЦ РУКОВОДСТВА.
-P пейджер
Назначить используемый пейджер. Это опция переназначает переменную окружения MANPAGER, которая в свою очередь переназначает переменную PAGER. По умолчанию, man использует /usr/bin/less -isR.
-S список_разделов
Список разделов руководства разделённых двоеточиями, в которых осуществляется поиск. Эта опция переопределяет переменную окружения MANSECT.
-a
По умолчанию после вывода первой найденной страницы руководства, man завершит работу. Применение это опции вынудит man показать не только первую, а все страницы справочника подходящие под заданное имя.
-c
Переформатировать исходную страницу руководства, даже если существует актуальная отформатированная страница. Это может понадобиться, если раньше страница была отформатирована для экрана с другим количеством колонок, или повреждена.
-d
В действительности не показывает страницы справочника, но печатает отладочную информацию как при осуществлении вывода страниц.
-D
Показывает и страницу и отладочную информацию.
-f
Равнозначно команде whatis.
-F или --preformat
Форматирование без отображения.
-h
Выводит справку по опциям командной строки и завершает работу.
-k
То же что и команда apropos.
-K
Поиск заданной строки во *всех* страницах справочника. Предупреждение: возможно этот процесс будет очень долгим! Здесь может помочь указание раздела. (Просто для приблизительной оценки, на моей машине это работает примерно 500 страниц в минуту.)
-m system
Задать для поиска альтернативный набор страниц справочника, находящийся на системе с указанным именем.
-p string
Назначить выполнение ряда препроцессоров перед nroff или troff. Не всё инсталляции имеют полный набор препроцессоров. Несколько препроцессоров и буквы для их обозначения: eqn (e), grap (g), pic (p), tbl (t), vgrind (v), refer (r).

Эта опция переопределяет переменную окружения MANROFFSEQ.
-t
Использует /usr/bin/groff -Tps -mandoc для форматирования страницы справочника, без вывода на stdout. Возможно перед печатью на принтере вывод /usr/bin/groff -Tps -mandoc понадобится пропустить через некоторые фильтры.
-w или --path
Не отображает страницы справочника, но печатает местонахождение(я) тех файлов, что были бы отформатированы и показаны. Если аргумент не задан: выводит (на stdout) список каталогов в которых man осуществляет поиск страниц руководства. Если manpath это ссылка на man, то "manpath" равноценно "man --path".
-W
Подобно -w, но печатает по одному имени файла на строку без дополнительной информации. Это полезно в командах shell, например, man -aW man | xargs ls -

man (от англ. manual — руководство) — команда Unix, предназначенная для форматирования и вывода справочных страниц.

Вместо того, чтобы отображать man-страницу целиком, вы можете вывести лишь ее краткое описание, используя команду whatis.

Формат страниц

Для удобства навигации, все man-страницы соответствуют единому стандартному формату. Вот список некоторых разделов, которые часто используются на страницах:

NAME — имя команды и краткое однострочное описание ее назначения.
SYNOPSIS — список опций и аргументов командной строки, которые принимает команда, либо параметры функции и ее заголовочный файл.
DESCRIPTION — более подробное описание назначения и принципов работы команды или функции.
EXAMPLES — типовые примеры использования, обычно от самых простых к более сложным.
OPTIONS — описания для каждой из опций, которые принимает команда.
EXIT STATUS — коды возврата и их значения.
FILES — связанные с командой или функцией файлы.
BUGS — вероятные проблемы, связанные с работой команды или функции и ожидающие решения. Также известны как KNOWN BUGS.
SEE ALSO — список связанных команд и функций.
AUTHOR, HISTORY, COPYRIGHT, LICENSE, WARRANTY — информация о программе: ее история, условия использования, создатели программы.

man -a keyword

выводит все доступные для keyword man-страницы

man -f keyword

ищет и выводит краткое описание всех man-страниц, где имеются ссылки на keyword

man --warnings

включает предупреждения

man -l ...

включает чувствительность к регистру

man -H[browser]

активирует вывод в HTML и просмотр в браузере, который определен в $BROWSER или определен по умолчанию во время компиляции (обычно lynx).

-k или --apropos, для поиска по ключевому слову в описаниях man-страниц.

man -k password

man --apropos password

С тем же успехом, вы также можете воспользоваться командой **apropos**:

apropos password

Если вы хотите произвести более углубленный поиск по всему содержимому страниц, используйте опцию -K:

man -K password

Поиск в man-странице вызывается с помощью клавишы "/", каждое последующее найденное вхождение можно просмотреть с помощью клавишы "n". Если вы знаете, что вам нужно, но не знаете точное название man-страницы, вам поможет apropos(1). Вы наверное обратили внимание на цифру в скобках после названия команды. Она задает категорию, к которой принадлежит команда. Для лучшей организации, и чтобы избежать дублирования, man-страницы делятся по категориям. Например, printf в Linux может принадлежать категориям 1, 1p, 3 и 3p. Она является как функцией библиотеки C, так и пользовательской командой, частью coreutils, которая часто используется в скриптах оболочки. В Linux категории, после номера которых следует "p", предназначены для POSIX-программистов. В таких случаях вы можете задать требуемую категорию, вставив ее номер между "man" и командой: man 3 printf. Ниже представлены категории, как они выглядят в современных системах Linux.

1 - исполняемые программы и команды оболочки;
2 - системные вызовы;
3 - библиотечные вызовы;
4 - файлы устройств (обычно расположены в /dev);
5 - форматы файлов;
6 - игры;

7 - макропакеты и соглашения;
8 - программы системного администрирования;
9 - процедуры ядра

CAT

Читает каждый файл из указанного списка и выводит содержимое на экран.

cat опции файл1 файл2 ...

Вы можете передать утилите несколько файлов и тогда их содержимое будет выведено поочередно, без разделителей.

Cat f

распечатывает содержимое файла f, а

cat f1 f2 > f3

сливает первые два файла и помещает результат в третий. Чтобы добавить файл f1 к файлу f2, надо выполнить команду

cat f1 >> f2

Если не указан ни один файл или среди аргументов встретился -, команда cat читает данные со стандартного ввода. Опции команды cat имеют следующий смысл:

-E - показывать символ $ в конце каждой строки;

-u Вывод не буферизуется (по умолчанию буферизуется);

-s Не сообщается о несуществующих файлах. (удалять пустые повторяющиеся строки);

-b - нумеровать только непустые строки;

-n - нумеровать все строки

-h - отобразить справку;

-v Визуализация непечатных символов (кроме табуляций, переводов строк и переходов к новой странице). Управляющие символы изображаются в виде ^X (CTRL+X); символ DEL (восьмеричное 0177) - в виде ^?. Символы, не входящие в набор ASCII (то есть со взведенным восьмым битом) выдаются в виде M-x, где x - определяемый младшими семью битами символ.

С опцией -v можно использовать следующие опции:

-t Визуализация символов табуляции в виде ^I.

-e Визуализация символов перевода строки в виде $ (строка при этом все же переводится).

Если опция -v не указана, то опции -t и -e игнорируются

ECHO

echo опции строка

Команда echo выдает на стандартный вывод свои аргументы, разделяя их пробелами и выдавая в конце символ перевода строки. Кроме того, поддерживаются следующие C-подобные соглашения о задании управляющих символов

-n - не выводить перевод строки;
-e - включить поддержку вывода Escape последовательностей;
-E - отключить интерпретацию Escape последовательностей.

Это все опции, если включена опция -e, то вы можете использовать такие Escape последовательности для вставки специальных символов (не забывайте об особой трактовке shell'ом символа \):

\b Как правило - символ "забоя"; на терминалах типа Dasher - перемещение в левый верхний угол экрана.
\c Не выдавать в конце символ перевода строки.
\f Переход к новой странице.
\n Перевод строки.
\r Возврат каретки.
\t Табуляция.
\v Вертикальная табуляция.
**** Сам символ ****.
\0n Здесь n - восьмеричный ASCII-код 8-битного символа, состоящий не более чем из трех цифр.

Вы можете разукрасить вывод echo с помощью последовательностей управления цветом Bash. Для доступны такие цвета текста

\033[30m - чёрный;
\033[31m - красный;
\033[32m - зелёный;
\033[33m - желтый;
\033[34m - синий;
\033[35m - фиолетовый;
\033[36m - голубой;
\033[37m - серый.

И такие цвета фона:

\033[40m - чёрный;
\033[41m - красный;
\033[42m - зелёный;
\033[43m - желтый;
\033[44m - синий;

\033[45m - фиолетовый;
\033[46m - голубой;
\033[47m - серый;
\033[0m - сбросить все до значений по умолчанию.

Вы можете вывести содержимое текущей папки просто подставив символ *

echo *

Также можно вывести файлы определенного расширения:

echo *.mkv

Вы можете использовать запись echo в файл linux:

echo 1 > file

TOUCH

Команда touch изменяет времена доступа и модификации всех файлов-аргументов. Если файл с указанным именем не существует, то он создается. Время задается в том же формате, что и для команды date(1):
ммддччмм[гг]
(месяц, день, часы, минуты, год). Если время не указано, то используется текущее время.

Опциям команды touch приписан следующий смысл:

-a Изменить только время последнего доступа;

-m Изменить только время последней модификации;

-c Не создавать файлы, если они не существуют;

-f Пытается обновить информацию о времени, даже если права доступа файла не позволяют делать.

-h Указывает утилите не изменять данные о файле, если он задан символической ссылкой.

-r file Использовать значения времени из файла, заданного аргументом file.

-t time Устанавливает время последнего изменения и доступа в соответствии с указанным форматом time.

Формат даты, указанный в ключе -t, задается в соответствии с шаблоном [[CC]YY]MMDDhhmm[.SS]:

CC — первые две цифры года (век).

YY — последние две цифры года.

Если параметр CC не задан и значение YY находится в пределах 69 и 99, то тогда CC устанавливается равным 19,

в противном случае используется 20.

MM — двузначный номер месяца.

DD — двузначный номер дня.

hh — значение часов даты.
mm — значение минут даты.
SS — значение секунд даты.

LS

Команда **ls** для каждого имени каталога распечатывает список входящих в этот каталог файлов; для файлов - повторяется имя файла и выводится дополнительная информация в соответствии с указанными флагами.

По умолчанию имена файлов выводятся в алфавитном порядке. Если имена не заданы, выдается содержимое текущего каталога. Если заданы несколько аргументов, то они сортируются по алфавиту, однако сначала всегда идут файлы, а потом каталоги с их содержимым.

Существует три основных формата выдачи. По умолчанию выдается по одному файлу в строке; флаги **-C** и **-x** позволяют выдавать информацию в несколько колонок, а флаг -m задает свободный формат. Для определения формата вывода при указании флагов **-C**, **-x** и **-m** используется переменная окружения COLUMNS, значение которой равно количеству символов в выходной строке. Если эта переменная не установлена, используется база данных terminfo(4) и значение переменной окружения TERM. Если эта информация недоступна, длина выходной строки берется равной 80.

Командой ls обрабатываются следующие флаги:

-R Рекурсивно обойти встретившиеся подкаталоги.

-a Вывести список всех файлов (обычно не выводятся файлы, имена которых начинаются с точки).

-d Если аргумент является каталогом, то выводить только его имя, а не содержимое. Часто используется с флагом **-l** для получения сведений о состоянии каталога.

-C Вывод в несколько колонок с сортировкой по колонкам.

-x Вывод в несколько колонок с сортировкой по строкам.

-m Вывод в свободном формате, имена файлов разделяются запятыми.

-l Вывод в длинном формате: перед именами файлов выдается режим доступа, количество ссылок на файл, имена владельца и группы, размер в байтах и время последней модификации (см. ниже). Если файл является специальным, то в поле размера выводится старший и младший номера устройства.

-n То же, что и **-l**, но идентификаторы владельца и группы выводятся в виде чисел, а не в виде имен.

-o То же, что и **-l**, но идентификатор группы не выводится.

-g То же, что и **-l**, но идентификатор владельца не выводится.

-r Изменить порядок сортировки на обратный алфавитный или, при наличии флага -t, сначала выводить более старые файлы.

-t Имена файлов сортируются не по алфавиту, а по времени (сначала идут самые свежие файлы). По умолчанию используется время последнего изменения. См. также флаги -u и -c.

-u Вместо времени последнего изменения использовать время последнего доступа для сортировки (с флагом -t) или для вывода (с флагом -l).

-c Вместо времени последнего изменения использовать время последней модификации описателя файла (т.е. время создания файла, изменения режима доступа к нему и т.п.) для сортировки (с флагом -t) или для вывода (с флагом -l).

-p Если файл является каталогом, то выдавать после его имени символ /.

-F Если файл является каталогом, то выдавать после его имени символ /; если файл является выполняемым, то выдавать после его имени символ *.

-b Выдавать непечатные символы, входящие в имя файла, в восьмеричном виде (\ddd).

-q Выдавать непечатные символы, входящие в имя файла, в виде символа ?.

-i Выдавать в первой колонке номера описателей файлов.

-s Выдавать размер файлов в блоках (включая косвенные блоки).

-f Рассматривать каждый аргумент как каталог и выводить его содержимое. Этот флаг отменяет флаги **-l, -t, -s, -r** и включает флаг **-a**. Сортировка имен файлов не производится; имена выдаются в таком порядке, в каком они перечислены в каталоге.

Режим доступа к файлу при указании флага -l выводится в виде 10 символов. Первый символ означает:

d Файл является каталогом.

b Файл является специальным блочным файлом.

c Файл является специальным символьным файлом.

p Файл является именованным каналом.

- Обычный файл.

Остальные 9 символов делятся на три группы по три символа: права доступа владельца, других пользователей из его группы, всех прочих пользователей. Внутри каждой группы используются три символа, обозначающие права на чтение, запись и выполнение файла соответственно. Для каталога под правом на выполнение подразумевается право на просмотр в поисках требуемого файла.

CD

Команда **cd** применяется для того, чтобы сделать заданный каталог текущим. Если каталог не указан, используется значение переменной окружения **$HOME** (обычно это каталог, в который Вы попадаете сразу после

входа в систему). Если каталог задан полным маршрутным именем, он становится текущим. Если маршрутное имя не полное, команда cd пытается найти каталог по одному из маршрутов, заданных переменной окружения **$CDPATH**. Способ задания и семантика этой переменной такие же, как у **$PATH**. По отношению к новому каталогу нужно иметь право на выполнение, которое в данном случае трактуется как разрешение на поиск.

Поскольку для выполнения каждой команды создается отдельный процесс, cd не может быть обычной командой; она распознается и выполняется shell'ом.

MORE

Этот фильтр позволяет осуществлять проверку текста, выведенного на один полный экран. После каждого заполнения экрана команда делает паузу и выводит на нижней строке экрана сообщение типа:

--More-

(дальше). Если при этом пользователь нажимает клавишу carriage return (возврат каретки), то на экран будет выведена следующая строка текста. Если пользователь нажимает клавишу SPACE (пробел), будет выведен следующий полный экран. Другие возможности описаны ниже.

ОПЦИИ

-n Целое число, используемое в качестве размера окна (в строках) вместо принятого для команды more по умолчанию.

-c Команда more отображает в верхней части экрана каждую страницу с начала, стирая при этом предварительно выведенную там строку. Это исключает прокрутку экрана, облегчая чтение текста с помощью команды more. Эта опция игнорируется, если терминал не имеет возможности очищать строку до конца.

-d Команда more в конце каждого заполненного экрана высвечивает сообщение " Hit space to continue, Rubout to abort" ("Нажмите пробел для продолжения, а Rubout -для прерывания"). Эта опция полезна, если команда more используется в качестве фильтра в некоторых системах, таких как класс, где многие пользователи могут быть неопытными.

-f Под действием этой опции команда more считает не экранные, а логические строки. То есть, длинные строки не заворачиваются. Рекомендуется использовать эту опцию, если вывод команды nroff подключается в конвейер через команду ul, которая может генерировать escape последовательности. Эти escape - последовательности содержат символы, которые обычно занимают экранные позиции, но не печатаются при выводе на терминал в качестве части escape - последовательности. Таким образом, команда more может решить, что строки длиннее, чем они есть на самом деле и ошибочно завернуть их.

-l Умышленно не обрабатывается команда Ctrl-L (form feed - переход на новую страницу). Если не задана эта опция, команда more останавливается после любой строки, содержащей Ctrl-L, до тех пор, пока экран не заполнится до конца. Также, если некоторый файл начинается со знака form feed, экран очищается прежде, чем распечатывается этот файл.

-u Обычно команда more осуществляет такое же подчеркивание, как при команде nroff, в том виде, который соответствует данному терминалу: если терминал может делать подчеркивание или имеет режим выделения, то выходные данные команды more соответствуют escape последовательностям для разрешения подчеркивания или режима выделения для подчеркнутого текста в исходном файле. Опция -u подавляет такую обработку.

-r Обычно, команда more игнорирует управляющие символы, которые она не интерпретирует некоторым образом. Под действием опции -r эти символы отображаются как ^C, где стоит "C" для любого такого символа.

-w Обычно, команда more завершает работу при подходе к концу своих входных данных. А под действием опции w, однако, команда more запрашивает пользователя и ждет нажатия любой клавиши перед завершением работы.

+linenumber Команда more стартует со строки с номером linenumber (номер строки).

+/pattern Команда more стартует просмотр текста за две строки до той строки, в которой содержится регулярное выражение pattern

GREP

(**g**lobal **r**egular **e**xpression **p**rint) - утилита для поиска и фильтрации текстовых данных на основе регулярных выражений

Утлита **grep** выполняет поиск *образца* в текстовых файлах и выдает все строки, содержащие этот образец. Она использует компактный недетерминированный алгоритм сопоставления.

Основной синтаксис:

cat <file name>|grep <string or regular expression>

<command>|grep <string or regular expression>

grep <string or regular expression> [file name]

Если **имя_файла** не указано, **grep** предполагает поиск в стандартном входном потоке. Обычно каждая найденная строка копируется в стандартный выходной поток. Если поиск осуществлялся в нескольких файлах, перед каждой найденной строкой выдается имя файла.

Для написания регулярного выражения используются помимо обычных символов спецсимволы:

- **+** — эквивалентно одному или нескольким вхождениям предыдущего символа.
- **^** начало строки
- **$** конец строки
- **?** — это означает хотя-бы одно повторение предыдущего символа. Так запись **«a?»** будет соответствовать **«a»** или **«aa».**
- **(**— начало выражения чередования.
- **)** — конец выражения чередования.
- **|** — сопоставление любого из выражений, разделенных символом '**|**'. Например: «**(a | b) cde»** будет соответствовать **«acde»,** либо **«bcde».**
- **{**— этот метасимвол указывает начало спецификатора диапазона. Например: «**a {2}**» соответствует вхождению **«aa»** в файле 2 раза.
- **}** — этот метасимвол указывает конец спецификатора диапазона.

В Basic Regular Expressions **(BRE)** метасимволы вроде: **'{', '}', '(', ')', '|', '+', '?'** теряют свой смысл и считаются нормальными символами строки и должны быть выделены специальным образом, если их следует рассматривать как специальные символы.

Будьте внимательны при использовании в **списке_образцов** символов **$**, *****, **[**, **^**, **|**, **(**, **)** и ****, поскольку они являются метасимволами командного интерпретатора. Лучше брать весь **список_образцов** в одиночные кавычки **'... '**.

Для того, чтобы команда воспринимала данные символы как специальные используется обратный слеш ****

-i игнорировать регистр букв в искомом регулярном выражении

-E Egrep или **grep -E** — это другая версия **grep** или **Extended grep.** Эта версия **grep** эффективна и быстра, когда дело доходит до поиска шаблона регулярных выражений, поскольку она обрабатывает метасимволы как есть и не заменяет их как строки. **Egrep** использует **ERE** или **Extended Regular Expression.** По сути позволяет использовать расширенные шаблоны

Утилита **/usr/bin/grep** использует для задания образцов *ограниченные регулярные выражения*

/usr/xpg4/bin/grep

Опции **-E** и **-F** влияют на способ интерпретации списка_образцов программой **/usr/xpg4/bin/grep**. Если указана опция **-E**, программа **/usr/xpg4/bin/grep** интерпретирует образцы в списке как полные регулярные выражения. Если же

указана опция **F**, **grep** интерпретирует **список_образцов** как фиксированные строки. Если ни одна из этих опций не указана, **grep** интерпретирует элементы **списка_образцов** как простые регулярные выражения

ОПЦИИ

Следующие опции поддерживаются обеими программами, **/usr/bin/grep** и **/usr/xpg4/bin/grep**:

-b	Предваряет каждую строку номером блока, в котором она была найдена. Это может пригодиться при поиске блоков по контексту (блоки нумеруются с 0).
-c	Выдает только количество строк, содержащих образец.
-h	Предотвращает выдачу имени файла, содержащего сопоставившуюся строку, перед собственно строкой. Используется при поиске по нескольким файлам.
-i	Игнорирует регистр символов при сравнениях.
-l	Выдает только имена файлов, содержащих сопоставившиеся строки, по одному в строке. Если образец найден в нескольких строках файла, имя файла не повторяется.
-n	Выдает перед каждой строкой ее номер в файле (строки нумеруются с 1).
-s	Подавляет выдачу сообщений о не существующих или недоступных для чтения файлах.
-v	Выдает все строки, за исключением содержащих образец.
-w	Ищет выражение как слово, как если бы оно было окружено метасимволами \< и \>.

Слово "грепать" входит в топ самых популярных терминов, используемых разработчиками. Оно происходит от одноимённой консольной утилиты grep (**g**lobal **r**egular **e**xpression **p**rint), выполняющей поиск по файлу или файлам определённого текста. Грепать для разработчиков — то же самое, что гуглить для тех, кто активно пользуется интернетом. Как правило, грепают файлы с исходным кодом или логи во время отладки.

Команда	Описание
grep pattern file.txt	поиск pattern в файле file.txt, с выводом полностью совпавшей строкой

grep -o pattern file.txt	поиск pattern в файле file.txt и вывод только совпавшего куска строки
grep -i pattern file.txt	игнорирование регистра при поиске
grep -bn pattern file.txt	показать строку (-n) и столбец (-b), где был найден pattern
grep -v pattern file.txt	инверсия поиска (найдет все строки, которые не совпадают с шаблоном pattern)
grep -A 3 pattern file.txt	вывод дополнительных трех строк, после совпавшей
grep -B 3 pattern file.txt	вывод дополнительных трех строк, перед совпавшей
grep -C 3 pattern file.txt	вывод три дополнительные строки перед и после совпавшей
grep -r pattern $HOME	рекурсивный поиск по директории $HOME и всем вложенным
grep -c pattern file.txt	подсчет совпадений
grep -L pattern *.txt	вывести список txt-файлов, которые не содержат pattern
grep -l pattern *.txt	вывести список txt-файлов, которые содержат pattern
grep -w pattern file.txt	совпадение только с полным словом pattern
grep -f patterns.txt file.txt	поиск по нескольким pattern из файла patterns.txt, шаблоны разделяются новой строкой

grep -I pattern file.txt	игнорирование бинарных файлов
grep -v -f file2 file1 > file3	вывод строк, которые есть в file1 и нет в file2
grep -in -e 'python' `find -type f`	рекурсивный поиск файлов, содержащих слово *python* с выводом номера строки и совпадений
grep -inc -e 'test' `find -type f` | grep -v :0	рекурсивный поиск файлов, содержащих слово *python* с выводом количества совпадений
grep . *.py	вывод содержимого всех py-файлов, предваряя каждую строку именем файла
grep "Http404" apps/**/*.py	рекурсивный поиск упоминаний *Http404* в директории apps в py-файлах

ШАБЛОНЫ

В shell реализована обработка определенного набора шаблонов подстановки, вот некоторые из них:

- ?: соответствует одному и только одному символу, независимо от того чем этот символ является;
- [...]: соответствует одному символу, найденному в скобках. Символы можно указать как диапазон символов (то есть 1-9) или *дискретные значения* , или даже и то и другое. Пример: [a-zBE5-7] будет соответствовать всем символам между a и z, B, E, 5, 6 или 7;
- [!...]: соответствует любому символу, *не* находящемуся в скобках. Например, [!a-z], будет соответствовать любому символу который не является буквой в нижнем регистре[5];
- {c1,c2}: соответствует c1 или c2, где c1 и c2 также шаблоны подстановки, которые обозначают что вы можете написать {[0-9]*,[acr]} например.

Вот некоторые шаблоны и их значения:

- /etc/*conf: все файлы в каталоге /etc с окончаниями в именах conf. Это может соответствовать /etc/inetd.conf, /etc/conf.linuxconf, *а также /etc/conf* если такой файл существует. Помните, что * может соответствовать пустой строке.
- image/{cars,space[0-9]}/*.jpg: все файлы, заканчивающиеся на .jpg в каталогах image/cars, image/space0, (...), image/space9, если эти каталоги существуют.
- /usr/share/doc/*/README: все файлы с именем README во всех каталогах /usr/share/doc непосредственно. Это будет соответствовать /usr/share/doc/mandrake/README, например, но не соответствовать /usr/share/doc/myprog/doc/README.
- *[!a-z]: все файлы, имена которых *не* заканчиваются буквой в нижнем регистре в текущем каталоге.

КАВЫЧКИ

Кавычки, обрамляющие строку, предотвращают интерпретацию специальных символов, которые могут находиться в строке. Символ называется "специальным", если он не только означает самого себя но и имеет дополнительное значение для программ, например символ шаблона -- *. (одинарные и двойные)

Многие программы в передаваемых им параметрах, используют специальные символы, в этом случае нужно заключать их в кавычки, что-бы системная оболочка их не трогала, оставляя для вызываемой программы.(одинарные).

grep '[Tt]*' ./[Bb]*

При обращении к переменным, желательно использовать двойные кавычки. Это позволит не интерпретировать специальные символы, содержащиеся в именах переменных, за исключением символов $, ` (обратная кавычка) и \ (обратный слэш). То что символ $ является исключением, позволяет производить подстановку переменных в строке.

Кроме вышесказанного, двойные кавычки используются для предотвращения разбиения строки на слова:

```
#!/usr/local/bin/bash

var="is a variable"
echo this $var # здесь команде передается  4 аргумента
echo "this $var"  # а тут один
```

Результат обеих команд echo, будет одинаков, но только на первый взгляд:

```
freebsd82 /work-tests/sh# ./test.sh
this is a variable # здесь на выходе мы получаем 4 слова
this is a variable # а здесь одну строку
```

Заключать в кавычки аргументы команды echo, нужно только если разбиение вывода на слова, вызывает какие-то трудности.

Обратные кавычки ` ` подставляют результат команды, которая написана между ними.

ПРОВЕДЕНИЕ СИМВОЛА “\”

Поведение символа "\" (обратный слэш), зависит от разных факторов, таких как: экранирован-ли он, заключен-ли в кавычки, используется в подстановке команд или в конструкции "вложенный документ".

```
#  Простое экранирование и кавычки
                # возвращаемое значение

echo \z             #  z
echo \\z            # \z
echo '\z'           # \z
echo '\\z'          # \\z
echo "\z"           # \z
echo "\\z"          # \z

#  Подстановка команды
echo `echo \z`      #  z
echo `echo \\z`     #  z
echo `echo \\\z`    # \z
echo `echo \\\\z`   # \z
echo `echo \\\\\\z` # \z
echo `echo \\\\\\\z` # \\z
echo `echo "\z"`    # \z
echo `echo "\\z"`   # \z

# Встроенный документ
cat <<EOF
\z
EOF                 # \z

cat <<EOF
```

\\z
EOF # \z

Больше про кавычки https://vds-admin.ru/shell-scripting/kavychki

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?

   cp 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?

SHELL заменяет тильду на абсолютный путь к домашней директории.

7) Как рассчитывается размер символьной ссылки?

Размер символьной ссылки равен (число символов в названии исходного файла) байт

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? (храняться в массиве, который объявляется в коде в параметрах main)

10) Любой вопрос про vi

11) Какие команды и символы обрабатываются shell, а какие нет? Как это определяется?

12) Сигналы

13) Переменные окружения

HOME содержит путь к домашнему каталогу текущего пользователя.

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.

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/

14) Откуда терминал знает, что нужно запустить файл, путь к которому не указан?
Пути до программ исполняемых файлов прописаны в переменной PATH.

15) Если существует программа и файл с одинаковыми именами, что запустится при вызове этого имени? (приоритет исполнения)
Запустится программа

17) ^C ^D. Нажатие ^D передает символ конца файла “EOF”. Имеет смысл при вводе, потому что означает конец ввода. ФФТо 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?

Кавычки, обрамляющие строку, предотвращают интерпретацию специальных символов, которые могут находиться в строке. Символ называется "специальным", если он не только означает самого себя но и имеет дополнительное значение для программ, например символ шаблона -- *. (одинарные и двойные)

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 * ?

SHELL’ом

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

Ей будет плохо (работать будет неправильно) (в ls есть ключ -b, который показывает непечатные знаки и записывает всё в строку, что исправляет проблему)
23) man поиск по кейвордам, главы, уметь пользоваться

24) как строки хранятся в памяти?
25) Как в man создать новую страницу?
https://www.opennet.ru/man.shtml?topic=man&category=7&russian=0
26) Уметь работать с more
27) Как передать отдельной программе какое-то конкретное значение переменной окружения?
Для этого необходимо присвоить значение необходимой переменной в строке перед вызовом программы
MANPATH=dir/man man sth

28) Можно ли создавать жёсткие ссылки на директории?

При создании директории автоматически создаются три жесткие ссылки, иначе их создавать запрещено системой. При создании жесткой ссылки на директорию возникнет неоднозначность при переходе в родительскую папку, так как их будет несколько, плюс может зациклиться рекурсивный поиск, поэтому создание жестких ссылок запрещено.