## Лабораторная работа№ 4. Работа с файловой системой

### Текущий каталог

Файловая система не только систематизирует данные, но и является основой метафоры "рабочего места" в Linux. Каждая выполняемая программа "работает" в строго определённом каталоге файловой системы. Такой каталог называется **текущим каталогом**, можно представлять, что программа во время работы "находится" именно в этом каталоге, это её "рабочее место". В зависимости от текущего каталога может меняться поведение программы: зачастую программа будет по умолчанию работать с файлами, расположенными именно в текущем каталоге -- до них она "дотянется" в первую очередь. Текущий каталог есть у любой программы, в том числе и у командной оболочки (shell) пользователя. Поскольку взаимодействие пользователя с системой обязательно опосредовано командной оболочкой, можно говорить о том, что пользователь "находится" в том каталоге, который в данный момент является *текущим каталогом его командной оболочки*.

Все команды, отдаваемые пользователем при помощи shell, наследуют текущий каталог shell, т. е. "работают" в том же каталоге. По этой причине пользователю важно знать текущий каталог shell. Для этого служит утилита `pwd`:

```
[methody@localhost methody]$ pwd
/home/methody
[methody@localhost methody]$
```

Текущий каталог: pwd

`pwd` (аббревиатура от print working directory) возвращает полный путь текущего каталога командной оболочки, естественно, именно той командной оболочки, при помощи которой была выполнена команда `pwd`. В данном случае Мефодий узнал, что в этот момент (на этой **виртуальной консоли**) текущим является каталог "`/home/methody`".

Почти все утилиты, с которыми работал Мефодий на предыдущих занятиях, по умолчанию читают и создают файлы в текущем каталоге. Так, Мефодий обращался к файлам, не используя никаких путей, просто по имени. Например, использовал утилиту cat, чтобы вывести на экран содержимое файла "text".

```
[methody@localhost methody]$ cat text
File: info.info,  Node: Help-Cross,  Up: Cross-refs

The node reached by the cross reference in Info
 . . .
[methody@localhost methody]$ cat /home/methody/text
File: info.info,  Node: Help-Cross,  Up: Cross-refs

The node reached by the cross reference in Info
 . . .
```

### Полный и относительный путь к файлу

В действительности, командная оболочка, прежде чем передавать параметр "text" (имя файла) утилите cat, подставляет значение текущего каталога -- получается полный путь к этому файлу в файловой системе: "/home/methody/text". Содержимое именно этого файла утилита cat выведет на экран[1]. Набирая только имя файла без пути к текущему каталогу, Мефодий воспользовался **относительным путём** к этому файлу.

| | |
|---|---|
| **относительный путь** | **Путь** к объекту **файловой системы**, не начинающийся в **корневом каталоге**. Для каждого **процесса** Linux определён **текущий каталог**, с которого система начинает относительный путь при выполнении файловых операций. |

Относительный путь строится точно так же, как и полный -- перечислением через "/" всех названий каталогов, встретившихся при движении к искомому каталогу или файлу. Между полным путём и относительным есть только одно существенное различие: относительный путь начинается *от текущего каталога*, в то время как полный путь всегда начинается *от корневого ката-*

[1] Вообще говоря, в нескольких разных каталогах файловой системы могут оказаться файлы с именем "text", именно поэтому командная оболочка всегда передаёт программам и утилитам "точный адрес" файла в файловой системе -- **полный путь**.

*лога*. Относительный путь любого файла или каталога в файловой системе может иметь любую конфигурацию: чтобы добраться до искомого файла можно двигаться как по направлению к корневому каталогу, так и от него. Linux различает полный и относительный пути очень просто: если имя объекта *начинается* на "/" -- это полный путь, в любом другом случае -- относительный.

Пользователь может обращаться к файлу при помощи полного или относительного пути -- результат будет совершенно тот же. Так, команды `cat text` и `cat /home/methody/text`, отданные Мефодием, дали одинаковый результат, поскольку выводился *один и тот же* файл. Если в относительном пути встречаются символы "/", рассматриваются *подкаталоги* текущего каталога, их подкаталоги и т. д. Короче говоря, **относительный путь** строится по тем же правилам, что и **полный**, с той разницей, что относительный путь начинается *не* с символа "/". Сам **текущий каталог**, каков бы ни был полный путь к нему, всегда имеет ещё одно обозначение, ".", которое можно использовать, если по каким-то причинам *требуется*, чтобы даже в *относительном* пути к файлу, находящемуся в *текущем* каталоге, присутствовал элемент "имя каталога". Так, пути "`text`" и "`./text`" тоже приводят к одному и тому же файлу, однако в первом случае в строке пути не содержится ничего, кроме имени файла.

Отделить путь к файлу от его имени можно с помощью команд `dirname` и `basebane` соответственно:

```
[methody@localhost methody]$ basename /home/methody/text
text
[methody@localhost methody]$ basename text
text
[methody@localhost methody]$ dirname /home/methody/text
/home/methody
[methody@localhost methody]$ dirname ./text
.
[methody@localhost methody]$ dirname text
.
```

Использование `dirname` и `basename`

Мефодий заметил, что для "`text`" и "`./text`" `dirname` выдало одинаковый результат: ".", что понятно: как было сказано выше, эти формы пути совершенно

эквивалентны, а при *автоматической* обработке результатов dirname гораздо лучше получить ".", чем *пустую строку*.

## Домашний каталог

Мефодий заметил, что на прошлых занятиях, заходя с разных виртуальных консолей по очереди и одновременно, он всегда оказывался в одном и том же **текущем каталоге**: он всё время обращался к своим файлам при помощи относительного пути и всегда находил нужные. Это не случайно -- в Linux у каждого пользователя обязательно есть *свой собственный* каталог, который и становится текущим сразу после **регистрации в системе** -- **домашний каталог**[2]. Для Мефодия домашним каталогом является "/home/methody".

| | |
|---|---|
| **домашний каталог** | Каталог, предназначенный для хранения собственных данных пользователя Linux. Как правило, является **текущим** непосредственно после регистрации пользователя в системе. **Полный путь** к домашнему каталогу хранится в **переменной окружения** HOME. |

Поскольку каждый пользователь располагает своим собственным каталогом и по умолчанию работает в нём, решается задача разделения файлов разных пользователей. Обычно доступ других пользователей к чужому домашнему каталогу ограничен: наиболее типична ситуация, когда пользователи могут читать содержимое файлов друг друга, но не имеют права их изменять или удалять.

## Информация о каталоге

Чтобы иметь возможность ориентироваться в файловой системе, нужно знать, что содержится в каждом каталоге. Запомнить всю структуру файловой системы невозможно и не нужно: в любой момент можно просмотреть содержимое любого каталога при помощи утилиты ls (сокращение от англ. "list" -- "список"):

---

[2] Домашний каталог указывается в **учётной записи** пользователя

```
[methody@localhost methody]$ ls
-filename-with-  text
[methody@localhost methody]$
```

Команда ls

Поданная без параметров, команда ls выводит список файлов и каталогов, содержащихся в *текущем каталоге*[3]. При помощи этой утилиты Мефодий обнаружил, что в его **домашнем каталоге** (который в данный момент является текущим) содержатся два файла, созданные утилитой info на позапрошлой лекции: "-filename-with-" и "text".

Утилита ls принимает один **параметр**: имя каталога, содержимое которого нужно вывести. Имя может быть задано любым доступным способом: в виде **полного** или **относительного пути**. Например, чтобы получить список в файлов в своём домашнем каталоге, Мефодий мог бы использовать команды "ls /home/methody" и "ls ." -- результат был бы аналогичным.

Кроме параметра, утилита ls "понимает" множество ключей, которые нужны главным образом для того, чтобы выводить дополнительную информацию о файлах в каталоге или выводить список файлов выборочно. Чтобы узнать обо всех возможностях ls, нужно, конечно же, прочесть **руководство** по этой утилите ("man ls").

Почитав руководство по ls, Мефодий решил изучить содержимое своей файловой системы и начал с начала -- с **корневого каталога**.

```
[methody@localhost methody]$ ls -F /
bin/   dev/  home/  mnt/   root/  swap/  tmp/  var/
boot/  etc/  lib/   proc/  sbin/  sys/   usr/
[methody@localhost methody]$
```

Команда ls -F

Мефодий использовал ключ -F, чтобы отличать файлы от каталогов. При наличии этого ключа ls в конце имени каждого каталога ставит символ "/", чтобы показать, что в нём может содержаться что-то ещё. В выведенном

[3] Вот пример утилиты, которая по умолчанию работает с файлами в текущем каталоге.

списке нет ни одного файла -- в корневом каталоге содержатся только подкаталоги.

Кроме того, Мефодий решил получить более подробную информацию о содержимом своего домашнего каталога:

```
[methody@localhost methody]$ ls -aF
-filename-with-  .bash history  .bashrc  .lpoptions  .rpmmacros    Documents/
./               .bash_logout   .emacs   .mutt/      .xemacs/      text
../              .bash_profile  .i18n    .pinerc     .xsession.d/  tmp/
[methody@localhost methody]$
```

Команда `ls -aF`

Внезапно он обнаружил, что файлов в его домашнем каталоге не два, а гораздо больше. Дело в том, что утилита `ls` по умолчанию не выводит информацию об объектах, чьё имя начинается с "." -- в том числе о "." и "..". Для того, чтобы посмотреть *полный* список содержимого каталога, и используется ключ "`-a`" (**a**ll)[4]. Как правило, с "." начинаются имена **конфигурационных файлов** и **конфигурационных каталогов**, работа с которыми (т. е. *настройка* окружения, "рабочего места") не пересекается с работой над какой-нибудь прикладной задачей (хотя, конечно, эффективность работы зависит от хорошо настроенного окружения). Кроме того, подобных файлов в домашнем каталоге активно работающего пользователя со временем заводится немало (по одному на каждую приличную утилиту) и их присутствие в выдаче `ls` сильно загромождает её.

Разберёмся подробно в списке файлов в домашнем каталоге Мефодия. Начнём с весьма лаконичных имён "." и "..". Мефодий уже знает, что "." -- это имя текущего каталога. Следующее имя в списке, ".." -- это ссылка на **родительский каталог**. Родительский каталог -- это тот каталог, *в котором* находится данный. Родительским каталогом для "`/home/methody`" будет каталог "`/home`": он получается просто отбрасыванием последнего имени каталога в полном

[4] Такое поведение `ls` напоминает принцип работы файловых менеджеров со скрытыми файлами в системах MS-DOS/Windows. Разница в том, что в MS-DOS/Windows скрытые файлы предусмотрены файловой системой -- файл может иметь *атрибут* "скрытый" и при этом называться как угодно. В Linux скрытые файлы -- это не свойство файловой системы, а только *соглашение по наименованию* файлов

пути. Иначе можно сказать, что родительский каталог -- это один шаг по дереву каталогов по направлению к корню. ".." -- это сокращённый способ сослаться на родительский каталог: пока текущим каталогом является "/home/methody", **относительный путь** ".." (или, что то же самое, "./..") будет эквивалентен "/home". С использованием ".." можно строить *сколь угодно* длинные пути, такие как "../../usr/../var/log/../run/../../home"

---

Не сразу понятно, что приводит этот путь всё туда же, в "/home".

---

, однако в действительности они применяются только при автоматической подстановке в программах, а во время работы пользователя необходимости в такого рода усложнениях не возникает.

| | |
|---|---|
| **родительский каталог** | Каталог, в котором содержится данный. Для **корневого каталога** родительским является он сам. |

Ссылки на текущий и на родительский каталог обязательно присутствуют в *каждом* каталоге в Linux. Даже если каталог пуст, т. е. не содержит ни одного файла или подкаталога, команда "ls -a" выведет список из двух имён: "." и "..".

За ссылками на текущий и родительский каталоги следуют несколько файлов и каталогов, имена которых начинаются на ".". В них содержатся настройки **командной оболочки** (файлы, начинающиеся с ".bash") и других программ. В домашнем каталоге каждого пользователя Linux всегда присутствует несколько таких файлов. Использование этих файлов позволяет пользователям независимо друг от друга настраивать поведение командной оболочки и других программ -- организовывать своё "рабочее место" в системе. Подробнее речь об этом пойдёт на следующих занятиях.

## Перемещение по дереву каталогов

Пользователь может работать с файлами не только в своём домашнем каталоге, но и в других каталогах. В этом случае будет удобно *сменить* **текущий каталог**, т. е. "переместиться" в другую точку файловой системы. Для смены текущего каталога командной оболочки используется команда `cd` (от англ. "change directory" -- "сменить каталог"). Команда `cd` принимает один параметр: имя каталога, в который нужно переместиться -- сделать текущим. Как обычно, в качестве имени каталога можно использовать полный или относительный путь.

```
[methody@localhost methody]$ cd /home
[methody@localhost home]$ ls
methody  shogun
[methody@localhost home]$ cd methody
[methody@localhost methody]$
```

Смена текущего каталога

Сначала Мефодий решил переместиться в каталог "`/home`", и посмотреть, что ещё есть в этом каталоге, кроме его домашнего каталога. Он обнаружил ещё один каталог -- "`shogun`", и догадался, что это домашний каталог Гуревича, **входное имя** которого -- "`shogun`". Кроме того, он заметил, что изменился вид **приглашения командной строки** (подсказки shell) -- слово "`methody`" заменилось на "`home`". В приглашении командной строки часто указывается текущий каталог shell -- чтобы пользователю легче было ориентироваться, в каком каталоге он "находится" в данный момент.

После этого Мефодий решил вернуться в свой домашний каталог, но в этом случае он использовал уже не полный, а относительный путь -- "`cd methody`". Вводя эту команду, Мефодий не стал целиком набирать имя своего домашнего каталога, а набрал только первые буквы "`me`" и нажал клавишу Tab, как ему советовал Гуревич. Командная оболочка умеет *достраивать* имена файлов и каталогов: пользователю достаточно набрать несколько первых символов

имени файла или каталога и нажать Tab. Если есть только один вариант завершения имени -- оболочка закончит его сама, и пользователю не придётся набирать оставшиеся символы. Достраивание -- весьма серьёзное средство экономии усилий и повышения эффективности при работе с командной строкой. Современные командные оболочки умеют достраивать имена файлов и каталогов, а также имена команд. Достраивание наиболее развито в командном интерпретаторе zsh.

Те же самые перемещения -- в родительский каталог и обратно -- Мефодий мог бы сделать и набирая значительно меньше символов. Для перемещения в родительский каталог ("/home") удобно воспользоваться ссылкой "..". Необходимость вернуться в домашний каталог из произвольной точки файловой системы возникает довольно часто, поэтому командная оболочка поддерживает обозначение домашнего каталога при помощи символа "~". Поэтому чтобы перейти в домашний каталог из любого другого, достаточно выполнить команду "cd ~". При исполнении команды символ "~" будет заменён командной оболочкой на полный путь к домашнему каталогу пользователя.

```
[methody@localhost methody]$ cd ..
[methody@localhost home]$ cd ~
[methody@localhost methody]$ cd ~shogun
[methody@localhost shogun]$ cd
[methody@localhost methody]$
```

Переход в родительский и в домашний каталог

При помощи символа "~" можно ссылаться и на домашние каталоги других пользователей: "*~имя пользователя*". В примере выше Мефодий перешёл в домашний каталог Гуревича командой "cd ~shogun". Команда cd, поданная без параметров, эквивалента команде "cd ~" и делает текущим каталогом домашний каталог пользователя.

## Создание каталогов

Пользователь, конечно, не должен хранить все свои файлы в одном каталоге. В домашнем каталоге пользователя, как и в любом другом, можно создавать сколь угодно много подкаталогов, в них -- свои подкаталоги и т. д. Другими словами, пользователю принадлежит фрагмент (поддерево) файловой системы, корнем которого является домашний каталог пользователя.

Чтобы организовать такое поддерево, потребуется создать каталоги внутри домашнего. Для этого используется утилита mkdir. Она используется с одним обязательным параметром: именем создаваемого каталога. По умолчанию каталог будет создан в текущем каталоге.

```
[methody@localhost methody]$ mkdir examples
[methody@localhost methody]$ ls -F
-filename-with-  Documents/  examples/  text  tmp/
[methody@localhost methody]$
```

Создание каталога

Мефодий решил навести некоторый порядок в своём домашнем каталоге и поместить все файлы с примерами и упражнениями в отдельном подкаталоге -- "examples". Теперь, создав каталог, нужно переместить в него все файлы с примерами.

## Копирование и перемещение файлов

Для перемещения файлов и каталогов предназначена утилита mv (сокращение от англ. "move" -- "перемещать"). У mv два обязательных параметра: первый -- перемещаемый файл или каталог, второй -- файл или каталог назначения. Имена файлов и каталогов могут быть заданы в любом допустимом виде: при помощи полного или относительного пути. Кроме того, mv позволяет перемещать не только один файл или каталог, а сразу несколько. За подробностями о допустимых параметрах и ключах следует обратиться к руководству по mv.

```
[methody@localhost methody]$ mv -- -filename-with- examples/
[methody@localhost methody]$ cd examples
[methody@localhost examples]$ mv ../text .
[methody@localhost examples]$ ls
-filename-with-  text
[methody@localhost examples]$
```

Перемещение файлов

Мефодий сначала переместил в каталог "examples" файл "-filename-with-", поскольку имя этого файла начинается с "-", ему потребовалось предварить его ключом "--", чтобы следующее слово было воспринято командной оболочкой как параметр (этот приём был описан ранее). Затем он перешёл в каталог "examples" и переместил из родительского каталога ("../") файл "text" в текущий каталог ("."). Теперь в каталоге "examples" два файла с примерами.

Перемещение файла внутри одной файловой системы в действительности равнозначно его *переименованию*: данные самого файла при этом остаются на тех же секторах диска, изменяются *каталоги*, в которых произошло перемещение. Перемещение предполагает удаление ссылки на файл из того каталога, откуда он перемещён, и добавление ссылки на этот самый файл в тот каталог, куда он перемещён. В результате изменяется полное имя файла -- **полный путь**, т. е. положение файла в файловой системе.

Иногда требуется создать копию файла: для бОльшей сохранности данных, для того, чтобы создать модифицированную версию файла и т. п. В Linux для этого предназначена утилита cp (сокращение от англ. "copy" -- "копировать"). Утилита cp требует присутствия двух обязательных параметров: первый -- копируемый файл или каталог, второй -- файл или каталог назначения. Как обычно, в именах файлов и каталогов можно использовать полные и относительные пути. Есть несколько возможностей при комбинации файлов и каталогов в параметрах cp -- о них можно прочесть в **руководстве**.

```
[methody@localhost examples]$ cp text text.bak
[methody@localhost examples]$ ls
-filename-with-  text  text.bak
```

Копирование файлов

Мефодий решил создать резервную копию файла "text", "text.bak" в том же каталоге, что и исходный файл. Для этой простейшей операции копирования достаточно передать cp в качестве двух параметров имя исходного файла и имя копии. По умолчанию cp, как и многие другие утилиты, будет работать с файлами в текущем каталоге.

Нужно иметь в виду, что в Linux утилита cp нередко настроена таким образом, что при попытке скопировать файл поверх уже существующего не выводится никакого предупреждения. В этом случае файл будет просто перезаписан, а данные, которые содержались в старой версии файла, бесповоротно потеряны. Поэтому при использовании cp следует всегда быть внимательным и проверять имена файлов, которые нужно скопировать.

Говоря о копировании, уместно вспомнить широко известное высказывание, приписываемое Уильяму Оккаму: "Не следует умножать сущности сверх необходимого". Созданная при помощи cp копия файла связана с оригиналом только в воспоминаниях пользователя, в файловой системе исходный файл и его копия -- две совершенно независимые и ничем не связанные единицы. Поэтому при наличии нескольких копий одного и того же файла в рамках *одной файловой системы* повышается вероятность запутаться в копиях или забыть о некоторых из них. Если задача состоит в том, чтобы обеспечить доступ к одному и тому же файлу из разных точек файловой системы, нужно использовать специально предназначенный для этого механизм файловой системы Linux -- ссылки.

## Файл и его имена: ссылки

### Жёсткие ссылки

Каждый файл представляет собой область данных на жёстком диске компьютера или на другом носителе информации, которую можно найти по *имени*. В файловой системе Linux содержимое файла связывается с его именем при помощи **жёстких ссылок**. Создание файла с помощью любой программы

означает, что будет создана жёсткая ссылка -- имя файла, и открыта новая область данных на диске. Причём количество ссылок на одну и ту же область данных (файл) не ограничено, т. е. у файла может быть несколько имён.

Пользователь Linux может добавить файлу ещё одно имя (создать ещё одну жёсткую ссылку на файл) при помощи утилиты ln сокращение от англ. "link" -- "соединять, связывать"). Первый параметр -- это имя файла, на который нужно создать ссылку, второй -- имя новой ссылки. По умолчанию ссылка будет создана в текущем каталоге.

```
[methody@localhost methody]$ ln examples/text text-hardlink
[methody@localhost methody]$ ls -lR
./:

. . .

drwxr-xr-x  3 methody methody 4096 Окт 16 04:45 examples
-rw-r--r--  2 methody methody  653 Окт  6 10:31 text-hardlink

./examples:
итого 92
-rw-r--r--  1 methody methody 84718 Окт  6 10:31 -filename-with-
-rw-r--r--  2 methody methody   653 Окт  6 10:31 text
```

Создание жёстких ссылок

Мефодий создал в своём домашнем каталоге жёсткую ссылку с именем "text-hardlink" на файл "text", который находится в подкаталоге "examples". Выведя подробный список файлов текущего каталога и его подкаталогов ("ls -lR"), Мефодий обратил внимание, что у файлов "text" и "text-hardlink" совпадают и размер ("653"), и время создания. Это его совершенно не удивило, поскольку он знает, что теперь "/home/methody/text-hardlink" и "/home/methody/examples/text" -- это два имени одного и того же файла. В подробном описании, выведенном командой "ls -l", Мефодию остались непонятны только два первых поля. Как объяснил Гуревич, первое "слово", состоящее из знаков "-drwx", -- это обозначение **прав доступа** к файлу, о которых речь пойдёт на следующих занятиях. А следующее за ним число -- количество жёстких ссылок на данный файл или каталог. У "text" и "text-hardlink" стоит число "2" -- у этого файла два имени.

Доступ к одному и тому же файлу при помощи нескольких имён может понадобиться в следующих случаях:

1. Одна и та же программа известна под несколькими именами.
2. Доступ пользователей к некоторым каталогам в системе может быть ограничен из соображений безопасности. Однако если всё же нужно организовать доступ пользователей к файлу, который находится в таком каталоге, можно создать жёсткую ссылку на этот файл в другом каталоге.
3. Современные файловые системы даже на домашних персональных компьютерах могут насчитывать до нескольких десятков тысяч файлов и тысячи каталогов. Обычно у таких файловых систем сложная многоуровневая иерархическая организация -- в результате пути ко многим файлам становятся очень длинными. Чтобы организовать более удобный доступ к файлу, который находится очень "глубоко" в иерархии каталогов, также можно использовать жёсткую ссылку в более доступном каталоге.
4. Полное имя некоторых программ может быть весьма длинным (например, i586-alt-linux-gcc-3.3), к таким программам удобнее обращаться при помощи сокращённого имени (жёсткой ссылки) -- gcc-3.3.

**Индексные дескрипторы**

Поскольку благодаря жёстким ссылкам у файла может быть несколько имён, понятно, что вся существенная информация о файле в файловой системе привязана не к имени. В файловых системах Linux вся информация, необходимая для работы с файлом, хранится в **индексном дескрипторе**. Для *каждого* файла существует индексный дескриптор: не только для обычных файлов, но

и для каталогов[5], **файлов-дырок** и т. д. Каждому файлу соответствует ровно *один* индексный дескриптор.

Индексный дескриптор -- это описание файла, в котором содержится:

- тип файла (обычный файл, каталог, файл-дырка и т. д.);
- **права доступа** к файлу;
- информация о том, кому принадлежит файл;
- отметки о времени создания, модификации, последнего доступа к файлу;
- размер файла;
- указатели на физические блоки на диске, принадлежащие этому файлу -- в этих блоках хранится "содержимое" файла.

Все индексные дескрипторы пронумерованы, поэтому номер индексного дескриптора -- это уникальный идентификатор файла в файловой системе -- в отличие от *имени* файла (жёсткой ссылки на него), которых может быть несколько. Узнать номер индексного дескриптора любого файла можно при помощи всё той же утилиты `ls` с ключом `-i`:

```
[methody@localhost methody]$ ls -i ./text-hardlink examples/text
127705 examples/text  127705 ./text-hardlink
```

Информация об индексных дескрипторах файлов

Мефодий решил поинтересоваться номерами индексных дескрипторов файла "`text`" и жёсткой ссылки на него "`text-hardlink`" -- он обнаружил, что эти номера совпадают ("`127705`"), то есть этим двум именам соответствует один индексный дескриптор, т. е. один и тот же файл.

Все операции с файловой системой -- создание, удаление и перемещение файлов -- производятся на самом деле над индексными дескрипторами, имена нужны только для того, чтобы пользователь мог легко ориентироваться в файловой системе. (Было бы очень неудобно запоминать многозначный номер каждого нужного файла или каталога.) Более того, имя (или имена) файла *не*

[5] Каталоги в Linux -- тоже файлы особого типа

*указаны* в его индексном дескрипторе. В файловой системе Ext2 имена файлов хранятся *в каталогах*: каждый каталог представляет собой список имён файлов и номеров их индексных дескрипторов. Жёсткую ссылку (имя файла, хранящееся в каталоге) можно представлять как каталожную карточку, на которой указан номер индексного дескриптора -- идентификатор файла.

| | |
|---|---|
| **жёсткая ссылка** | Запись вида *имя файла+номер индексного дескриптора* в **каталоге**. Жёсткие ссылки в Linux -- основной способ обратиться к файлу *по имени*. |

### Символьные ссылки

У жёстких ссылок есть два существенных ограничения:

1. Жёсткая ссылка может указывать только на файл, но не каталог, потому что в противном случае в файловой системе могут возникнуть циклы -- бесконечные пути.
2. Жёсткая ссылка не может указывать на файл на другой файловой системе. Например, невозможно создать на жёстком диске жёсткую ссылку на файл, расположенный на дискете[6].

Чтобы избежать этих ограничений, были разработаны **символьные ссылки**. Символьная ссылка -- это просто файл, в котором содержится имя другого файла. Символьные ссылки, как и жёсткие, предоставляют возможность обращаться к одному и тому же файлу по разным именам. Кроме того, символьные ссылки могут указывать и на каталог, чего не позволяют жёсткие ссылки. Символьные ссылки называются так потому, что содержат *символы* -- путь к файлу или каталогу.

| | |
|---|---|
| **символьная ссылка** | Файл особого типа ("l"), в котором содержится путь к другому файлу. Если на пути к файлу встречается символьная ссылка, система выполняет *подстановку*: исходный путь заменяется на тот, что содержится в ссылке. |

[6] Причина этого ограничения в том, что номер индексного дескриптора уникален только в рамках одной файловой системы. В разных файловых системах могут оказаться два разных файла с одинаковым номером индексного дескриптора, в результате будет невозможно установить, на какой из них указывает жёсткая ссылка

Символьную ссылку можно создать при помощи команды ln с ключом "-s" (сокращение от "symbolic"):

```
[methody@localhost methody]$ ln -s examples/text text-symlink
[methody@localhost methody]$ ls -li

. . .

127699 drwxr-xr-x  2 methody methody 4096 Окт  4 17:12 examples
127705 -rw-r--r--  2 methody methody  653 Сен 30 10:04 text-hardlink
  3621 lrwxrwxrwx  1 methody methody   13 Окт  4 18:05 text-symlink -> exam-
ples/text
[methody@localhost methody]$
```

Создание символьных ссылок

Теперь Мефодий решил создать в своём домашнем каталоге и символьную ссылку на файл text и назвать её text-symlink. Команда ls -li отобразила этот файл совсем не так, как остальные: стрелочка ("->") указывает, куда направлена ссылка. Кроме того, Мефодий обратил внимание, что номер индексного дескриптора (первое поле), размер и время создания файла text-symlink отличаются text-hardlink, а также во втором поле (количество жёстких ссылок на файл) text-symlink указано "1". Все эти признаки недвусмысленно свидетельствуют о том, что text-symlink и text -- это *разные* файлы. Однако если выполнить команду cat text-symlink, то на экран будет выведено содержимое файла text.

Символьная ссылка вполне может содержать имя несуществующего файла, в этом случае ссылка будет существовать, но не будет "работать": например, если попробовать вывести содержимое такой "битой" ссылки при помощи команды cat, будет выдано сообщение об ошибке.

Узнать, куда указывает символьная ссылка, можно при помощи утилиты realpath:

```
[methody@localhost methody]$ realpath text-symlink
/home/methody/examples/text
```

Раскрытие символьных ссылок

## Удаление файлов и каталогов

В Linux для удаления файлов предназначена утилита rm (сокращение от англ. "remove" -- "удалять").

```
[methody@localhost methody]$ rm examples/text
[methody@localhost methody]$ ls -l text-hardlink
-rw-r--r--  1 methody methody 653 Сен 30 10:04 text-hardlink
[methody@localhost methody]$ rm text-hardlink
[methody@localhost methody]$ ls -l text-hardlink
ls: text-hardlink: No such file or directory
```

Удаление файла

Разобравшись в ссылках, Мефодий решил удалить файл text в каталоге examples. После этого файл text-hardlink в домашнем каталоге Мефодия, который является жёсткой ссылкой на удалённый файл text продолжает благополучно существовать. Единственное отличие, которое заметил Мефодий -- количество жёстких ссылок на этот файл теперь уменьшилось с "2" до "1" -- действительно, text-hardlink -- теперь единственное имя этого файла. Получается, что Мефодий удалил только одно из имён этого файла (**жёсткую ссылку**), сам файл остался нетронутым.

Однако если Мефодий удалит и жёсткую ссылку text-hardlink -- у этого файла больше не останется ни одного имени, он станет недоступным пользователю файловой системы и будет уничтожен.

Утилита rm предназначена именно для удаления жёстких ссылок, а не самих файлов. В Linux, чтобы полностью удалить файл, требуется последовательно удалить все жёсткие ссылки на него. При этом все жёсткие ссылки на файл (его имена) равноправны -- среди них нет "главной", с исчезновением которой исчезнет файл. Пока есть хоть одна ссылка, файл продолжает существовать. Впрочем, у большинства файлов в Linux есть только одно имя (одна жёсткая ссылка на файл), поэтому команда rm *имя файла* успешно удалит файл в большинстве случаев.

Как уже говорилось, символьные ссылки -- это отдельные файлы, поэтому после того, как Мефодий удалил файл text, text-symlink, который ссылался

на этот файл, продолжает существовать, однако теперь это -- "битая ссылка", поэтому его также можно удалить командой rm.

Мефодий решил создать каталог для разных упражнений -- test, а потом решил обойтись одним каталогом examples. Однако команда rm не сработала, заявив, что test -- это каталог:

```
[methody@localhost methody]$ mkdir test
[methody@localhost methody]$ rm test
rm: невозможно удалить `test': Is a directory
[methody@localhost methody]$ rmdir test
[methody@localhost methody]$
```

Удаление каталога

Для удаления *каталогов* предназначена другая утилита -- rmdir (от англ. "remove directory"). Впрочем, rmdir согласится удалить каталог только в том случае, если он пуст: в нём нет никаких файлов и подкаталогов. Удалить каталог вместе со всем его содержимым можно командой rm с ключом "-r" (recursive). Команда rm -r *каталог* -- очень удобный способ потерять в одночасье *все* файлы: она рекурсивно[7] обходит весь *каталог*, удаляя всё, что попадётся: файлы, подкаталоги, символьные ссылки... а ключ "-f" (force) делает её работу ещё неотвратимее, так как подавляет запросы вида "удалить защищённый от записи файл", так что rm работает безмолвно и безостановочно.

---

Помните: если вы удалили файл, значит, он уже не нужен, и не подлежит восстановлению!

---

В Linux не предусмотрено процедуры восстановления удалённых файлов и каталогов. Поэтому стоит быть *очень* внимательным, отдавая команду rm и, тем более, rm -r: нет никакой гарантии, что удастся восстановить случайно удалённые данные. Узнав об этом, Мефодий не огорчился, но подумал, что

[7] "Рекурсивно" по отношению к каталогам обозначает, что действие будет произведено над самим каталогом, его подкаталогами, подкаталогами его подкаталогов и т. д.

впредь будет удалять только *действительно* ненужные файлы, а всё сомнительное -- перемещать с помощью mv в подкаталог ~/tmp, где оно не будет мозолить глаза, и где можно периодически наводить порядок.

## КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Назначение утилиты pwd
2. Определение относительного пути
3. Отличие относительного и полного пути. Как различить полный и относительный путь
4. Для чего используется обозначение «.»?
5. Определение термина «домашний каталог»
6. Назначение команды ls. Назначение ключей команды ls
7. Определение понятия «родительский каталог»
8. Назначение и параметры команды cd. Применение клавиши Tab.
9. Команда для создания каталогов
10. Команды для копирования и перемещения файлов.
11. Понятие «жесткая ссылка». Порядок создания жесткой ссылки.
12. Необходимость использования нескольких имен для доступа к файлу
13. Что такое индексный дескриптор
14. Порядок создания индексных дескрипторов
15. Ограничения жестких ссылок
16. Определение символьной ссылки
17. Порядок создания символьной ссылки
18. Порядок удаления файлов и каталогов
19. Можно ли восстановить удаленные файлы и каталоги средствами ОС Linux?