Лабораторная работа №2. Терминал Linux

Как стало понятно из предыдущего занятия (**Сеанс работы в Linux**), основное средство общения с Linux -- системная клавиатура и экран монитора, работающий в **текстовом режиме**. Вводимый пользователем текст немедленно отображается на мониторе соответствующими буквами, однако может и не отображаться, как в случае ввода пароля. Для *управления* вводом используются некоторые *нетекстовые* клавиши на клавиатуре: *Backspace* (он же "Забой") -- для удаления последнего введённого символа или *Enter* -- для передачи команды системе. Нажатие на эти клавиши не приводит к отображению символа, вместо этого вводимый текст обрабатывается системой тем или иным способом.

```
[methody@localhost methody]$ data
 -bash: data: command not found
[methody@localhost methody]$ date
 Вск Сен 12 13:59:36 MSD 2004
```

Сообщение об ошибке

Вначале Мефодий ошибся, и вместо команды date написал data. В ответ он получил **сообщение об ошибке**, поскольку такой команды система не понимает. Затем (этого не видно в примере, но случилось именно так!) он *снова* набрал data, но вовремя одумался и, нажав клавишу *Backspace*, удалил последнее "a", вместо которого ввёл "e", превратив data в date. Такая команда в системе есть, и на экране возникла текущая дата.

Диалог машины и пользователя неспроста выглядит как обмен текстами. Именно письменную речь используют люди для постановки и описания решения задач в заранее определённом, формализованном виде. Поэтому и задача управления системой может целиком быть представлена и решена в виде формализованного текста -- программы. При этом машине отводится роль аккуратного исполнителя этой программы, а человеку -- роль автора. Кроме того, человек анализирует текст, получаемый от системы: запрошенную им информацию и т. н. **сообщения** -- текст, описывающий состояние системы в процессе решения задачи (например, сообщение об ошибке "command not found").

Текстовый принцип работы с машиной позволяет отвлечься от конкретных частей компьютера, вроде системной клавиатуры и видеокарты с монитором, рассматривая единое *оконечное устройство*, посредством которого пользователь *вводит* текст и передаёт его системе, а система *выводит* необходимые пользователю данные и сообщения. Такое устройство называется **терминалом**. В общем случае терминал -- это *точка входа* пользователя в систему, обладающая способностью передавать текстовую информацию. Терминалом может быть отдельное **внешнее устройство**, подключаемое к компьютеру через

порт последовательной передачи данных (в персональном компьютере он называется "COM port"). В роли терминала может работать (с некоторой поддержкой со стороны системы) и программа (например, xterm или ssh). Наконец, **виртуальные консоли** Linux -- тоже терминалы, только организованные программно с помощью подходящих устройств современного компьютера.

Терминал	Устройство последовательного ввода и вывода символьной информации, способное воспринимать часть символов как **управляющие** для редактирования ввода, посылки **сигналов** и т. п. Используется для взаимодействия пользователя и системы

Для приёма и передачи текста терминалу достаточно уметь принимать и передавать *символы*, из которых этот текст состоит. Более того, *желательно* чтобы единицей обмена с компьютером был именно один байт (один acsii-символ). Тогда каждая буква, набранная на клавиатуре, может быть передана системе для обработки, если это понадобится. С другой стороны, типичный способ управления системой в Linux -- работа в **командной строке** -- требует *построчного* режима работы, когда набранный текст передаётся компьютеру только после нажатия клавиши *Enter* (что соответствует символу конца строки). Размер такой строки в байтах предугадать, конечно, нельзя, поэтому терминал, работающий в построчном режиме, ничем, по сути, не отличается от терминала, работающего в посимвольном режиме -- за исключением того, что активность системы по обработке приходящих с этого терминала данных падает в несколько раз (обмен ведётся не байтами, а целыми строками).

Свойство терминала передавать только символьную информацию приводит к тому, что некоторые из передаваемых символов должны восприниматься не как текстовые, а как **управляющие** (например, символы, возвращаемые клавишами *Backspace* и *Enter*). На самом деле управляющих символов больше: часть из них предназначены для экстренной передачи команд системе, часть -- для редактирования вводимого текста. Многие из этих символов не имеют *специальной* клавиши на клавиатуре, поэтому их необходимо извлекать с помощью **клавиатурного модификатора** *Ctrl*.

Команды, подаваемые с клавиатуры с помощью *Ctrl*, как и символы, передаваемые при этом системе, принято обозначать знаком "^", после которого следует имя клавиши, нажимаемой вместе с *Ctrl*: например, одновременное нажатие *Ctrl* и "a" обозначается "^A".

Так, для завершения работы программы `cat`, которая считывает построчно данные с клавиатуры и выводит их на терминал, можно воспользоваться командой "^C" или "^D":

```
[methody@localhost methody]$ cat
```

```
Any Text
 Any Text
^C

[methody@localhost methody]$ cat
Any Text agaim^[[Dn
 Any Text again
^D
[methody@localhost methody]$
```

Как завершить работу cat?

Одну строчку вида "Any Tex..." Мефодий вводит с клавиатуры (что отображается на экране), и после того, как Мефодий нажмёт *Enter*, она немедленно выводится программой `cat` (что тоже отображается на экране). С каждой последующей строкой программа `cat` поступила бы аналогично, но в примере Мефодий оба раза завершил работу программы, в первом случае нажав "^C", а во втором -- "^D". Эффект команды оказали одинаковый, а работают они по-разному: "^C" посылает программе, которая считывает с клавиатуры, сигнал аварийного прекращения работы, а "^D" сообщает ей, что ввод данных с клавиатуры закончен, можно продолжать работу дальше (поскольку программа `cat` больше ничего не делает, она завершается самостоятельно естественным путём). Можно считать, что "^C" -- это сокращение от "Cancel", а "^D" -- от "Done".

В пример не попало, как, набирая первый `cat`, Мефодий вновь ошибся и написал `ccat` вместо `cat`. Чтобы исправить положение, он воспользовался клавишами со стрелочками: с помощью клавиши *Стрелка влево* подвёл курсор к одному из "c" и нажал *Backspace*, а затем *Enter*. В режиме ввода *команды* это ему удалось, а при передаче данных программе `cat` клавиша *Стрелка влево* не сдвинула курсор, а передала целую последовательность символов: "^[", "[" и "D". Дело в том, что на клавиатуре терминала может быть так много разных нетекстовых клавиш, что на всех них не хватает ограниченного количества *разных* управляющих символов. Поэтому большинство нетекстовых клавиш возвращают т. н. **управляющую последовательность**, которая начинается управляющим символом (как правило -- Escape, т. е. "^["), за которым следует строго определённое число обычных (для клавиши *Стрелка влево* -- "[" и "D").

То же самое можно сказать и о *выводе* управляющих последовательностей на терминал. Современный терминал умеет довольно много помимо простого вывода текста: перемещать курсор по всему экрану (чтобы вывести текст там), удалять и вставлять строки на экране, использовать цвет и т. п. Всем этим заведуют **управляющие последовательности**, которые при выводе на экран терминала не отображаются как текст, а выполняются заранее

заданным способом. В некоторых случаях управляющие последовательности, возвращаемые клавишами, совпадают с теми, что управляют поведением терминала. Поэтому-то Мефодий и не увидел "Any Text again^[[Dn" в выдаче `cat`: "^[[D" при выводе на терминал перемещает курсор на одну позицию влево, так что было выведено "Any Text again", затем курсор встал прямо над "n" и поверх него было выведено "n". Если бы терминал имел вместо дисплея печатающее устройство, в этом месте обнаружилось бы нечто, состоящее из начертаний "n" и "n"[1].

Интерфейс командной строки. Взаимодействие пользователя с компьютером посредством терминала.

Требования к терминалу как к точке входа пользователя в систему получаются весьма невысокими. Формально говоря, терминал должен удовлетворять трём обязательным требованиям и одному необязательному. Терминал должен уметь:

1. передавать текстовые данные от пользователя системе;
2. передавать от пользователя системе немногочисленные управляющие команды;
3. передавать текстовые данные от системы пользователю;
4. (необязательно) интерпретировать некоторые данные, передаваемые от системы пользователю, как управляющие последовательности и соответственно обрабатывать их.

Ограничения на интерфейс напрямую не сказываются на эффективности работы пользователя в системе. Однако, чем меньше требований к интерфейсу, тем важнее разумно его организовать. Любое взаимодействие может быть описано с трёх точек зрения: во-первых, какую задачу решает пользователь (*что* он хочет от системы); во-вторых, *как* он формулирует задачу в доступном пониманию системы виде; и, в-третьих, какими средствами он пользуется при взаимодействии с системой. В частности, текстовый интерфейс удобно рассматривать с точки зрения предоставляемого им *языка* общения с машиной: во-первых, описанием этого языка задаётся диапазон решаемых с его помощью задач, а во-вторых,

[1] Некоторые терминалы умеют и так. Следует ещё иметь в виду, что терминалы разных типов имеют *разные* управляющие последовательности.

слова этого компьютерного языка (называемые в программировании *операторами*) предоставляют способ решения пользовательских задач (в виде небольших программ-сценариев). Команды, помогающие пользователю быстро и эффективно обмениваться с машиной предложениями на этом языке, и будут третьей составляющей интерфейса командной строки.

Командная строка

Основная среда взаимодействия с Linux -- **командная строка**. Суть её в том, что каждая строка, передаваемая пользователем системе, -- это **команда**, которую та должна выполнить. Пока не нажат *Enter*, строку можно редактировать, затем она отсылается системе.

```
[methody@localhost methody]$ cal
     Сентября 2004
 Вс Пн Вт Ср Чт Пт Сб
            1  2  3  4
  5  6  7  8  9 10 11
 12 13 14 15 16 17 18
 19 20 21 22 23 24 25
 26 27 28 29 30

[methody@localhost methody]$ echo Hello, world!
Hello, world!
```

Команды echo и cal

Команда `cal` выводит календарь на текущий месяц, а команда `echo` просто выводит на терминал всё, что следовало в командной строке *после неё*. Получается, что одну и ту же команду можно использовать с разными **параметрами** (или *аргументами*), причём параметры эти изменяют поведение команды. Здесь Мефодий захотел посмотреть календарь за март 2005-го года, для чего и передал команде cal два параметра -- 3 и 2005:

```
[methody@localhost methody]$ cal 3 2005
     Марта 2005
 Вс Пн Вт Ср Чт Пт Сб
         1  2  3  4  5
  6  7  8  9 10 11 12
 13 14 15 16 17 18 19
 20 21 22 23 24 25 26
 27 28 29 30 31
```

Команда `cal` с параметрами

В большинстве случаев при разборе командной строки *первое* слово считается именем команды, а остальные -- её параметрами. Более подробно о разборе командной строки и работе с ней рассказано ниже и на следующих занятиях.

Подсистема помощи

Пока же Мефодий решил, что узнал о командной строке достаточно для того, чтобы воспользоваться *главными* командами Linux (по частоте их употребления при изучении системы) -- `man` и `info`.

Работать с Linux, не заглядывая в документацию, практически невозможно. На это способны только седые аксакалы, у которых все нужные знания не то что в голове -- в кончиках пальцев, и предубеждённые новички. Всем прочим *настоятельно* рекомендуется, приступая к работе, а тем более -- к изучению Linux, пользоваться всеми доступными руководствами.

Все **утилиты**, все **демоны** Linux, все **функции ядра и библиотек**, структура большинства **конфигурационных файлов**, наконец, многие умозрительные, но важные понятия системы описаны либо в руководствах, либо в info-страницах, либо, на худой конец, в несистематизированной сопроводительной документации. Поэтому от пользователя системы не требуется *заучивать* все возможные варианты взаимодействия с ней. Достаточно *понимать* основные принципы её устройства и уметь находить справочную информацию. Эйнштейн говорил на этот счёт так: "Зачем запоминать то, что всегда можно посмотреть в справочнике?".

Страницы руководства (man)

Больше всего *различной* полезной информации содержится в **страницах руководства** (**manpages**). Каждая страница руководства (для краткости -- просто "руководство") посвящена какому-нибудь одному объекту системы. Для того, чтобы посмотреть страницу руководства, нужно дать команду системе man *объект*:

```
[methody@localhost methody]$ man cal
 CAL(1)          BSD General Commands Manual                 CAL(1)

 NAME
     cal - displays a calendar

 SYNOPSIS
     cal [-smjy13] [[month] year]

 DESCRIPTION
     Cal displays a simple calendar.  If arguments are not speci-
     fied, the current month is displayed.  The options are as
     follows:
. . .
```

Просмотр страницы руководства

"Страница руководства" занимает, как правило, больше одной страницы *экрана*. Для того, чтобы читать было удобнее, `man` запускает программу постраничного просмотра текстов -- `less`. Управлять программой `less` просто: страницы перелистываются пробелом, а когда читать надоест, надо нажать "q" (Quit). Перелистывать страницы можно и клавишами *Page Up*/*Page Down*, для сдвига на *одну* строку вперёд можно применять *Enter* или стрелку вниз, а на одну строку назад -- стрелку вверх. Переход на начало и конец текста выполняется по командам "g" и "G" соответственно (Go). Полный список того, что можно делать с текстом в `less`, выводится по команде "H" (Help).

Страница руководства состоит из **полей** -- стандартных разделов, с разных сторон описывающих заинтересовавший Мефодия объект -- команду `cal`. В поле NAME содержится краткое описание объекта (такое, чтобы его назначение было понятно с первого взгляда). В поле SYNOPSIS даётся *формализованное* описание способов использования объекта (с данном случае -- того, как и с какими параметрами запускать команду `cal`). Как правило, в квадратные скобки в этом поле заключены *необязательные* параметры команды, которые можно ей передать, а можно и опустить. Например, строка "[[month] year]" означает, что в это месте командной строки параметров у команды может не быть вообще, может быть указан год или пара -- месяц и год. Наконец, текст в поле DESCRIPTION -- это развёрнутое описание объекта, достаточное для того, чтобы им воспользоваться.

Одно из самых важных полей руководства находится в конце текста. Если в процессе чтения NAME или DESCRIPTION пользователь понимает, что не нашёл в руководстве того, что искал, он может захотеть посмотреть, а есть ли *другие* руководства или иные источники информации *по той же теме*. Список таких источников содержится в поле SEE ALSO:

```
[methody@localhost methody]$ man man
 . . .
 SEE ALSO
        apropos(1), whatis(1), less(1), groff(1), man.conf(5).
 . . .
```

Поле SEE ALSO руководства

До этого поля Мефодий добрался с помощью уже известной команды "G". Не то, чтобы ему неинтересно было читать руководство по `man`, скорее наоборот: им двигала любознательность. В Поле SEE ALSO обнаружились ссылки на руководства по `less`, `groff` (программе форматирования страницы руководства), структуре **конфи-**

гурационного файла для man, а также по двум сопутствующим командам с такими говорящими названиями[2], что Мефодий немедленно применяет одну команду к имени другой, не заглядывая даже в документацию. Так *ни в коем случае* не следует делать! А что если запущенная Вами программа начнёт с того, что сотрёт все файлы в Вашем каталоге?

```
[methody@localhost methody]$ whatis apropos
 apropos              (1)  - search the whatis database for strings
[methody@localhost methody]$ man apropos
 apropos(1)                              apropos(1)

 NAME
       apropos - search the whatis database for strings
 . . .
```

Вызов `whatis`

На этот раз Мефодию повезло: команда `whatis` не делает ничего разрушительного. Как и команда `apropos`, `whatis` ищет подстроку в некоторой базе данных, состоящей из полей NAME всех страниц помощи в системе. Различие между ними -- в том, что `whatis` ищет только среди имён объектов (в *левых* частях полей NAME), а `apropos` -- по всей базе. В результате у `whatis` получается список кратких описаний объектов с *именами*, включающими в себя искомое слово, а у `apropos` -- список, в котором это слово *упоминается*. Для того, чтобы это узнать, всё равно пришлось один раз прочесть документацию.

В системе может встретиться несколько объектов *разного* типа, но с одинаковым названием. Часто совпадают, например, имена системных вызовов (функций ядра) и программ, которые позволяют пользоваться этими функциями из командной строки (т. н. утилит).

```
[methody@localhost methody]$ whatis passwd
 passwd               (1)  - update a user's authentication tokens(s)
 passwd               (5)  - password file
 passwd               (8)  - manual page for passwd wrapper version 1.0.5
```

Руководства с одинаковыми именами

Описания объектов, выводимые `whatis`, отличаются числом в скобках -- номером раздела. В системе руководств Linux девять разделов, каждый из которых содержит страницы руководства к объектам определённого типа. Все разделы содержат по одному

[2] "Apropos" по-французски означает "кстати", а "whatis" -- по-английски -- "чтотакое".

руководству с именем "intro", в котором в общем виде и на примерах рассказано, что за объекты имеют отношение к данному разделу:

```
george@localhost:~> whatis intro
intro                 (1)  - Introduction to user commands
intro                 (2)  - Introduction to system calls
intro                 (3)  - Introduction to library functions
intro                 (4)  - Introduction to special files
intro                 (5)  - Introduction to file formats
intro                 (6)  - Introduction to games
intro                     (7)  - Introduction to conventions and
miscellany section
intro                   (8)  - Introduction to administration and
privileged commands
intro                 (9)  - Introduction to kernel interface
```

Руководства intro

Вот названия разделов в переводе на русский:

1. Команды пользователя.
2. Системные вызовы (пользовательские функции ядра Linux; руководства рассчитаны на программиста, знающего язык Си).
3. Библиотечные функции (функции, принадлежащие всевозможным библиотекам подпрограмм; руководства рассчитаны на программиста, знающего язык Си).
4. Внешние устройства и работа с ними (в Linux они называются специальными файлами).
5. Форматы различных стандартных файлов системы (например, конфигурационных).
6. Игры, безделушки и прочие вещи, не имеющие системной ценности.
7. Теоретические положения, договорённости и всё, что не может быть классифицировано.
8. Инструменты администратора (часто недоступные обычному пользователю).
9. Интерфейс ядра (*внутренние* функции и структуры данных ядра Linux, необходимы только системному программисту, исправляющему или дополняющему ядро).

В частности, пример с `passwd` показывает, что в системе "Some Linux", которую использует Мефодий, есть программа `passwd` (именно с её помощью Мефодий поменял себе пароль на прошлом занятии), *файл* `passwd`, содержащий информацию о пользователях и *администраторская* программа `passwd`, обладающая более широкими возможностями.

По умолчанию man просматривает все разделы и показывает *первое найденное* руководство с заданным именем. Чтобы посмотреть руководство по объекту из определённого раздела, необходимо в качестве первого параметра команды man указать номер раздела:

```
[methody@localhost methody]$ man 1 passwd
 PASSWD(8)     System Administration Utilities           PASSWD(8)
 . . .
[methody@localhost methody]$ man -a passwd
 PASSWD(1)    Some Linux                                 PASSWD(1)
 . . .
 PASSWD(8)    System Administration Utilities           PASSWD(8)
 . . .
 PASSWD(5)     Linux Programmer's Manual                 PASSWD(5)
 . . .
```

Выбор среди страниц руководства с одинаковым именем

Если в качестве первого параметра man использовать "-a", будут последовательно выданы *все* руководства с заданным именем. *Внутри* страниц руководства принято непосредственно после имени объекта ставить в круглых скобках номер раздела, в котором содержится руководство по этому объекту: man(1), less(1), passwd(5) и т. д.

Info

Другой источник информации о Linux и составляющих его программах -- справочная подсистема info. Страница руководства, несмотря на обилие ссылок *различного* типа, остаётся "линейным" текстом, структурированным только логически. Документ info структурирован прежде всего *топологически* -- это настоящий гипертекст, в котором множество небольших страниц объединены в дерево. В каждом разделе документа info всегда есть оглавление, из которого можно перейти сразу к нужному подразделу, откуда всегда можно вернуться обратно. Кроме того, info-документ можно читать и как *непрерывный* текст, поэтому в каждом подразделе есть ссылки на предыдущий и последующий подразделы.

```
[methody@localhost methody]$ info info
 File: info.info,  Node: Top,  Next: Getting Started,  Up: (dir)

 Info: An Introduction
 . . .
 * Menu:

 * Getting Started::             Getting started using an Info reader.
 * Expert Info::                 Info commands for experts.
 * Creating an Info File::       How to make your own Info file.
 * Index::                       An index of topics, commands, and variables.
 . . .
 --zz-Info: (info.info.bz2)Top, строк: 24 --All-----------------------------
 Welcome to Info version 4.6. Type ? for help, m for menu item.
```

Просмотр info-документа

Программа info использует весь экран: на большей его части она показывает текст документа, а первая и две последних строки отведены для ориентации в его структуре.

Одна или несколько страниц, которые можно перелистывать клавишей *Пробел* или *Page Up*/*Page Down* -- это **узел** (**node**). Узел содержит обычный текст и **меню** (**menu**) -- список ссылок на другие узлы, лежащие в дереве на более низком уровне. Ссылки внутри документа имеют вид "* *имя_узла*::" и перемещать по ним курсор можно клавишей *Tab*, а переходить к просмотру выбранного узла -- клавишей *Enter*. Вернуться к предыдущему просмотренному узлу можно клавишей "l" (от "Last"). И, главное, выйти из программы info можно, нажав "q" (Quit). Более подробную справку об управлении программой info можно в любой момент получить у самой `info`, нажав "?".

Узлы, составляющие документ `info`, можно просматривать и подряд, один за другим (с помощью команд "n", Next, и "p", Previous), однако это бывает нужно нечасто. В верхней строке экрана `info` показывает имя текущего узла, имя следующего узла и имя родительского (или верхнего) узла, в котором находится ссылка на текущий. Показанные Мефодию имя узла Top и имя верхнего узла (dir) означают, что просматривается *корневой* узел документа, выше которого -- только каталог со списком всех info-деревьев. В нижней части экрана расположена строка с информацией о текущем узле, а за ней -- строка для ввода длинных команд (например, для поиска текста с помощью команды "/").

Команде `info` можно указывать в параметрах всю *цепочку* узлов, приводящую к тому или иному разделу документации, однако это бывает нужно довольно редко:

```
[methody@localhost methody]$ info info "Getting Started" Help-Q
 File: info.info,  Node: Help-Q,  Prev: Help-Int,  Up: Getting
Started

 Quitting Info
 . . .
```

Просмотр определённого узла info-документа

Сам ли Мефодий это придумал, или подсказал кто, но совершенно правильно было заключить в кавычки имя узла "Getting Started" -- в этом случае info искала узел по "адресу" "info -> Getting Started -> Help-Q". Если бы команда имела вид `info info Getting Started Help-Q`, то "адрес" получился бы неправильный: "info -> Getting -> Started -> Help-Q". Ничего таинственного в этом нет, и уже к концу занятия станет понятно, в чём здесь дело.

RTFM

Оказывается, использование кавычек Мефодий придумал не сам: спросил у товарища, опытного пользователя Linux по фамилии Гуревич. Гуревич охотно показал, где ставить кавычки, а вот объяснять, что они делают, отказался, и заявил: "Там отличное руководство! Читай!". Документация в Linux играет важнейшую роль. Решение *любой* задачи должно начинаться с изучения руководств. Не стоит жалеть на это времени. Даже если рядом есть опытный пользователь Linux, который, возможно, *знает* ответ, не стоит беспокоить его *сразу же*. Вполне возможно, что, даже зная, *что* нужно сделать, он не помнит *как* именно -- и поэтому (а также потому, что он -- опытный пользователь) начнёт с изучения руководства. Это -- закон, у которого есть даже собственное название: **RTFM**, что означает "Read That Fine Manual"

RTFM	Правило, согласно которому решение любой задачи надо начинать с изучения документации.

Слова Гуревича -- практически дословный перевод этой фразы, так что её смысл и происхождение очевидны. Linux рассчитан в основном на тех, кто хочет знать, как им пользоваться.

Руководство -- это совсем не учебник, это -- справочник. В нём содержится информация, *достаточная* для освоения описываемого объекта, но никаких *обучающих* приёмов, никаких определений, повторений и выделения главного в нём обычно нет. Тем более не допускается *усечение* руководства с целью представить небольшие по объёму, но наиболее важные сведения. Так принято в учебниках, причём главные сведения раскрываются и объясняются очень подробно, а остальные присутствуют в виде ссылки на документацию для профессионалов. Страницы руководств -- и есть эта самая документация для профессионалов. Руководство чаще всего читает человек, который уже знает, о чём оно.

Это не значит, что из руководства нельзя понять, как, например, пользоваться командой в простейших случаях. Напротив, часто встречается поле EXAMPLES, которое как раз и содержит *примеры* использования команды в разных условиях. Однако всё это преследует цель не *научить*, а раскрыть смысл, пояснить сказанное в других полях. Мефодий нашёл описание работы двойных кавычек в руководстве по sh, однако понял из него далеко не всё -- главным образом, потому, что встретил слишком много незнакомых терминов.

Система `info` может содержать больше, чем `man`, поэтому в неё часто включают и учебники (принято называть учебник термином "tutorial"), и т. н. "howto" (примеры постановки и решения типовых задач), и даже статьи по теме. Таким образом info-документ может стать, в отличие от страницы руководства, *полным сводом* сведений. Разработка такого документа -- дело трудоёмкое, поэтому далеко не все объекты системы им сопровождаются.

Кроме того, и прочесть большой info-документ *целиком* зачастую невозможно. Поэтому имеет смысл начинать именно с руководства, а если его недостаточно -- изучать `info`.

Если некоторый объект системы не имеет документации ни в формате man, ни в формате info, это нехорошо. В этом случае можно надеяться, что при нём есть *сопроводительная документация*, не имеющая, увы, ни стандартного формата, ни тем более -- ссылок на руководства по другим объектам системы. Такая документация (равно как и примеры использования объекта), обычно помещается в каталог /usr/share/doc/*имя_объекта*.

Документация в подавляющем большинстве случаев пишется на простом английском языке. Если английский -- не родной язык для автора документации, она будет только проще. Традиция писать по-английски идёт от немалого вклада США в развитие компьютерной науки вообще и Linux в частности. Кроме того, английский становится языком международного общения во всех областях, не только в компьютерной. Необходимость писать на языке, который будет более или менее понятен большинству пользователей, объясняется постоянным развитием Linux. Дело не в том, что страницу руководства нельзя перевести, а в том, что её придётся переводить *всякий раз*, когда изменится описываемый ею объект! Например, выход новой версии программного продукта сопровождается изменением его возможностей и особенностей работы, а следовательно, и новой версией документации. Тогда *перевод* этой документации превращается в "moving target", сизифов труд.

Ключи

Работая в системе и изучая руководства, Мефодий заметил, что параметры команд можно отнести к двум различным категориям. Некоторые параметры имеют *собственный* смысл: это имена файлов, названия разделов и объектов в `man` и `info`, числа и т. п. Другие параметры собственного смысла не имеют, их значение можно истолковать, лишь зная, к какой команде они относятся. Например, параметр "-a" можно передать не только команде `man`, но и команде `who`, и команде `last`, при этом значить для них он будет разное. Такого рода параметры называются *модификаторами выполнения* или ключами (options).

```
[methody@localhost methody]$ date
Вск Сен 19 23:01:17 MSD 2004
[methody@localhost methody]$ date -u
Вск Сен 19 19:01:19 UTC 2004
```

Команда date с ключом

Для решения разных задач одни и те же действия необходимо выполнять слегка по-разному. Например, для синхронизации работ в разных точках земного шара лучше использовать единое для всех время (по Гринвичу), а для организации собственного рабочего дня -- местное время (с учётом сдвига по часовому поясу и разницы зимнего и летнего времени). И то, и другое время показывает команда date, только для работы по Гринвичу ей нужен дополнительный параметр-ключ "-u" (он же "--universal").

Однобуквенные ключи

Для формата ключей нет жёсткого стандарта, однако существуют договорённости, нарушать которые в наше время уже неприлично. Во-первых, если параметр начинается на "-", это -- **однобуквенный ключ**. За "-", как правило, следует один символ, чаще всего -- буква, обозначающая действие или свойство, которое этот ключ придаёт команде. Так проще отличать ключи от других параметров -- и пользователю при наборе командной строки, и программисту, автору команды.

Во-вторых, желательно, чтобы имя ключа было *значащим* -- как правило, это первая буква названия пресловутого действия или свойства. Например, ключ "-a" в `man` и `who` происходит от слова "**A**ll" (всё), и изменяет работу этих команд так, что они начинают показывать информацию, о которой они обычно умалчивают. А в командах cal и who смысл ключа "-m" -- разный:

```
[methody@localhost methody]$ who -m
 methody  tty1         Sep 20 13:56 (localhost)
[methody@localhost methody]$ cal -m
     Сентября 2004
 Пн Вт Ср Чт Пт Сб Вс
        1  2  3  4  5
  6  7  8  9 10 11 12
 13 14 15 16 17 18 19
 20 21 22 23 24 25 26
 27 28 29 30
```

Использование ключа "-m" в разных командах

Для `who` ключ "-m" означает "**M**e", то есть "Я", и в результате `who` работает похоже на whoami[3]. А для cal ключ "-m" -- это команда выдать календарь, считая первым днём *по-недельник* ("**M**onday"), как это принято в России.

[3] Кстати, с незапамятных времён who поддерживает один *нестандартный* набор параметров: who am i делает то же, что и who -m.

Свойство ключа быть, с одной стороны, предельно коротким, а с другой стороны -- информативным, называется **аббревиативностью**. Не только ключи, но и имена наиболее распространённых команд Linux обладают этим свойством.

В-третьих, иногда ключ изменяет поведение команды таким образом, что меняется и толкование параметра, следующего в командной строке за этим ключом. Выглядит это так, будто ключ *сам* получает параметр, поэтому ключи такого вида называются **параметрическими**. Как правило, их параметры -- имена файлов различного применения, числовые характеристики и прочие *значения*, которые нужно передать команде.

```
[methody@localhost methody]$ info info -o text
info: Запись ноды (info.info.bz2)Help-Cross...
info: Завершено.
[methody@localhost methody]$ cat text -n
     1  File: info.info,  Node: Help-Cross,  Up: Cross-refs
     2
     3  The node reached by the cross reference in Info
     4  ------------------------------------------------
 . . .
```

Использование info -o

Здесь `info` запустилась не в качестве *интерактивной* программы, а как обработчик info-документа. Результат работы программа поместила в файл text. А программа `cat` вывела содержимое этого файла на терминал, пронумеровав все строки (по просьбе ключа "-n", "number").

Теперь стало более-менее понятно, что означают неудобочитаемые строки в поле SYNOPSIS руководства. Например [-smjy13] из руководства по `cal` говорит о том, что команду можно запускать с необязательными ключами "-s", "-m", "-j", "-y", "-1" и "-3".

В-четвёртых, есть некоторые менее жёсткие, но популярные договорённости о *значении* ключей. Ключ "-h" ("**H**elp") обычно (но, увы, не всегда) заставляет команды выдать *краткую справку* (нечто похожее на SYNOPSIS, иногда с короткими пояснениями). Если указать "-" вместо имени *выходного* файла в соответствующем параметрическом ключе (нередко это ключ "-o"), вывод будет производиться на терминал[4]. Наконец, бывает необходимо передать команде *параметр, а не ключ*, начинающийся с "-". Для этого нужно использовать ключ "--":

[4] Точнее, на стандартный вывод

```
[methody@localhost methody]$ info -o -filename-with-
info: Запись ноды (dir)Top...
info: Завершено.
[methody@localhost methody]$ head -1 -filename-with-
head: invalid option -- f
Попробуйте `head --help' для получения более подробного описания.
[methody@localhost methody]$ head -1 -- -filename-with-
File: dir       Node: Top       This is the top of the INFO tree
```

Параметр-не ключ, начинающийся на "-"

Здесь Мефодий снова запустил `info` в качестве обработчика, который создал файл -filename-with-. Потом он пытался посмотреть первую строку этого файла (команда head -*количество_строк имя_файла* выводит первые *количество_строк* из указанного файла), но ничего не получилось, потому что имя файла начинается на "-", и программа head посчитала -f в начале имени ключом. Параметр "--" (первый "-" -- признак ключа, второй -- сам ключ) обычно запрещает команде интерпретировать все последующие параметры командной строки как ключи, независимо от того, начинаются ли они на "-" или нет. Только после "--" head согласилась с тем, что -filename-with- -- это имя файла.

Полнословные ключи

Аббревиативность ключей трудно соблюсти, когда их у команды *слишком* много. Некоторые буквы латинского алфавита (например, "s" или "o") используются очень часто, и могли бы служить сокращением сразу нескольких команд, а некоторые (например, "z") -- редко, под них и название-то осмысленное трудно придумать. На такой случай существует другой, **полнословный** формат: ключ начинается на *два* знака "-", за которыми следует *полное* имя обозначаемой им сущности. Таков, например, ключ "--help" (аналог "-h"):

```
[methody@localhost methody]$ head --help
Использование: head [КЛЮЧ]... [ФАЙЛ]...
Print the first 10 lines of each FILE to standard output.
With more than one FILE, precede each with a header giving the file name.
With no FILE, or when FILE is -, read standard input.

Аргументы, обязательные для длинных ключей, обязательны и для коротких.
  -c, --bytes=[-]N         print the first N bytes of each file;
                             with the leading `-', print all but the last
                             N bytes of each file
  -n, --lines=[-]N         print the first N lines instead of the first 10;
                             with the leading `-', print all but the last
                             N lines of each file
  -q, --quiet, --silent    не печатать заголовки с именами файлов
  -v, --verbose            всегда печатать заголовки с именами файлов
      --help     показать эту справку и выйти
      --version  показать информацию о версии и выйти


N may have a multiplier suffix: b 512, k 1024, m 1024*1024.
```

```
Об ошибках сообщайте по адресу <bug-coreutils@gnu.org>.
```

Ключ--help

Мефодий сделал то, о чём просила его утилита `head`. Видно, что некоторые ключи `head` имеют и однобуквенный, и полнословный формат, а некоторые -- только полнословный. Так обычно и бывает: часто используемые ключи имеют аббревиатуру, а редкие -- нет. *Значения* параметрических *полнословных* ключей принято передавать не следующим параметром командной строки, а с помощью конструкции "=*значение*" непосредственно после ключа.

Интерпретатор командной строки (shell)

В Linux нет отдельного объекта под именем "система". Система -- она на то и система, чтобы состоять из многочисленных компонентов, взаимодействующих друг с другом. Главный из системных компонентов -- пользователь. Это он командует машиной, а та его команды выполняет. В руководствах второго и третьего разделов описаны системные вызовы (функции ядра) и библиотечные функции. Они-то и есть непосредственные команды системе. Правда, воспользоваться ими можно только написав программу (чаще всего -- на языке Си), нередко -- программу довольно сложную. Дело в том, что функции ядра реализуют низкоуровневые операции, и для решения даже самой простой задачи пользователя необходимо выполнить несколько таких операций, преобразуя результат работы одной для нужд другой. Возникает необходимость выдумать для пользователя другой -- более высокоуровневый и более удобный в использовании -- язык управления системой. Все команды, которые использовал Мефодий в работе, были частью именно этого языка.

Из чего несложно было заключить, что *обрабатывать* эти команды, превращать их в последовательность системных и библиотечных вызовов должна тоже какая-нибудь специальная программа, и именно с ней непрерывно ведёт диалог пользователь сразу после входа в систему. Так оно и оказалось: программа эта называется **интерпретатор командной строки** или **командная оболочка** ("shell"). "Оболочкой" она названа как раз потому, что всё управление системой идёт как бы "изнутри" неё: пользователь общается с нею на удобном ему языке (с помощью текстовой командной строки), а она общается с другими частями системы на удобном *им* языке (вызывая запрограммированные функции).

Таким образом, упомянутые выше правила разбора командной строки -- это правила, действующие именно в командном интерпретаторе: пользователь вводит с терминала строку, shell считывает её, иногда -- преобразует по определённым правилам, получившуюся строку разбивает на команду и параметры, а затем выполняет команду, передавая ей

эти параметры. Команда, в свою очередь, анализирует параметры, выделяет среди них ключи, и делает что попросили, попутно выводя на терминал данные для пользователя, после чего завершается. По завершении команды возобновляется работа "отступившего на задний план" командного интерпретатора, он снова считывает командную строку, разбирает её, вызывает команду... Так продолжается до тех пор, пока пользователь не скомандует оболочке *завершиться* самой (с помощью logout или управляющего символа "^D", который для shell значит то же, что и для других программ: больше с терминала ввода не будет).

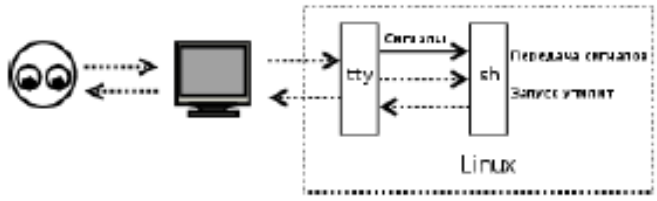
Конечно, командных интерпретаторов в Linux несколько. Самый простой из них, появившийся в ранних версиях UNIX, назывался sh, или "Bourne Shell" -- по имени автора, Стивена Борна (Stephen Bourne). Со временем его -- везде, где только можно -- заменили на более мощный, bash, "Bourne Again Shell"[5]. bash превосходит sh во всём, особенно в возможностях *редактирования* командной строки. Помимо sh и bash в системе может быть установлен "The Z Shell", zsh, *самый* мощный на сегодняшний день командный интерпретатор (шутка ли, 22 тысячи строк документации), или tcsh, обновлённая и тоже очень мощная версия старой оболочки "C Shell", синтаксис команд которой похож на язык программирования Си.

Когда Гуревич добавлял учётную запись Мефодия в систему, он не стал спрашивать того, какой командный интерпретатор ему нужен, потому что знал: для новичка имя командного интерпретатора -- пустой звук. Тем не менее имя оболочки, запускаемой для пользователя сразу после входа в систему -- т. н. **стартовый командный интерпретатор** (login shell), -- это часть пользовательской учётной записи, которую пользователь может изменить командой `chsh` (**ch**ange **sh**ell).

Какая бы задача, связанная с управлением системой, ни встала перед пользователем Linux, она должна иметь решение в терминах командного интерпретатора. Фактически, решение пользовательской задачи -- это описание её на языке shell. Язык общения пользователя и командного интерпретатора -- это высокоуровневый язык программирования, дополненный, с одной стороны, средствами организации взаимодействия команд и системы, а с другой стороны -- средствами взаимодействия с пользователем, облегчающими и ускоряющими работу с командной строкой.

---

[5] Игра слов: "Bourne Again" вслух читается как "born again", т. е. "возрождённый".



Интерфейс командной строки. Издание второе, переработанное и дополненное. Взаимодействие пользователя с компьютером посредством терминала и оболочки.

Команды и утилиты

```
[methody@localhost methody]$ apropos s
 . . . (четыре с половиной тысячи строк!)
```

Бессмысленная команда

Одного неудачного запуска apropos Мефодию было достаточно для того, чтобы понять: команд в Linux *очень* много. Ему пришло в голову, что никакая программа -- пусть даже и оболочка -- не может *самостоятельно* разбираться во всех задокументированных командах. Кроме того, Гуревич называл большинство команд **утилитами**, то есть полезными программами. Стало быть, командный интерпретатор не обязан уметь *выполнять* всё, что вводит пользователь. Ему достаточно разобрать командную строку, выделить из неё команду и параметры, а затем запустить утилиту -- программу, имя которой совпадает с именем команды.

В действительности *собственных* команд в командном интерпретаторе немного. В основном это -- операторы языка программирования и прочие средства управления самим интерпретатором. Все знакомые Мефодию команды, даже echo, существуют в Linux в виде отдельных утилит. shell занимается только тем, что подготавливает набор параметров в командной строке (например, раскрывая **шаблоны**), запускает программы и обрабатывает результаты их работы.

```
[methody@localhost methody]$ type info
 info is /usr/bin/info
[methody@localhost methody]$ type echo
 echo is a shell builtin
[methody@localhost methody]$ type -a echo
 echo is a shell builtin
 echo is /bin/echo
[methody@localhost methody]$ type -a -t echo
 builtin
 file
[methody@localhost methody]$ type -a -t date
 file
[methody@localhost methody]$ type -at cat
 file
```

Определение типа команды

В bash тип команды можно определить с помощью команды `type`. Собственные команды bash называются **builtin** (встроенная команда), а для утилит выводится **путь**, содержащий название каталога, в котором лежит файл с соответствующей программой, и имя этой программы. Некоторые -- самые нужные -- команды встроены в bash, даже несмотря на то, что они имеются в виде утилит (например, echo). Работает встроенная команда так же, но так как времени на её выполнение уходит существенно меньше, командный интерпретатор выберет именно её, если будет такая возможность. Ключ "-a" ("all", конечно), заставляет type вывести *все* возможные варианты интерпретации команды, а ключ "-t" -- вывести тип команды вместо пути.

По совету Гуревича Мефодий сгруппировал ключи, написав "-at" вместо "-a -t". Многие утилиты позволяют так делать, уменьшая длину командной строки. Если встречается параметрический ключ, он должен быть последним в группе, а его значение -- следовать, как и полагается, после. Группировать можно только однобуквенные ключи.

Слова и разделители

При разборе командной строки shell использует понятие **разделитель** (delimiter). Разделитель -- это символ, разделяющий слова; таким образом командная строка -- это последовательность *слов* (которые имеют значение) и *разделителей* (которые значения не

имеют). Для shell разделителями являются символ пробела, символ табуляции и символ перевода строки (который всё-таки может попасть *между* словами способом, описанным на следующих занятиях). Количество разделителей между двумя соседними словами значения не имеет.

Первое слово в тройке передаётся команде как первый параметр, второе -- как второй и т. д. Для того, чтобы разделитель попал *внутрь* слова (и получившаяся строка с разделителем передалась как *один* параметр), всю нужную подстроку надо окружить одинарными или двойными кавычками:

```
[methody@localhost methody]$ echo One      Two      Three
 One Two Three
[methody@localhost methody]$ echo One      "Two      Three"
 One Two      Three
[methody@localhost methody]$ echo 'One
>
> Ой. И что дальше?
> А, кавычки забыл!'
 One

 Ой. И что дальше?
 А, кавычки забыл!
[methody@localhost methody]$
```

Закавычивание в командной строке

В первом случае команде echo было передано *три* параметра -- "One", "Two" и "Three". Она их и вывела, разделяя пробелом. Во втором случае параметров было *два*: "One" и "Two Three". В результате эти *два* параметра были также выведены через пробел. В третьем случае параметр был всего *один* -- от открывающего апострофа "'One" до закрывающего "...забыл!'". Всё время ввода bash услужливо выдавал Мефодию подсказку "> " -- в знак того, что набор командной строки продолжается, но в режиме ввода содержимого кавычек.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Определение терминала
2. Понятие, назначение и обозначение управляющих символов
3. Назначение и отличие команд "^C" и "^D".
4. Определение управляющей последовательности
5. Требования к терминалу
6. Основной способ взаимодействия пользователя с Linux
7. Назначение, особенности использования команды `man`

8. Назначение, особенности использования команды `apropos`
9. Назначение, особенности использования команды `whatis`
10. Назначение ключа –a для команды `man`
11. Назначение, особенности использования команды `info`
12. Назначение и формат однобуквенных ключей
13. Необходимость использования ключа “--“
14. Необходимость и особенность использования полнословного формата ключей.
15. Определение интерпретатора командной строки
16. Правила группировки ключей
17. Слова и разделители командной строки