

Краткий мануал по консольным командам в Linux

Илья Дубков

Специально для
IoT школы SAMSUNG

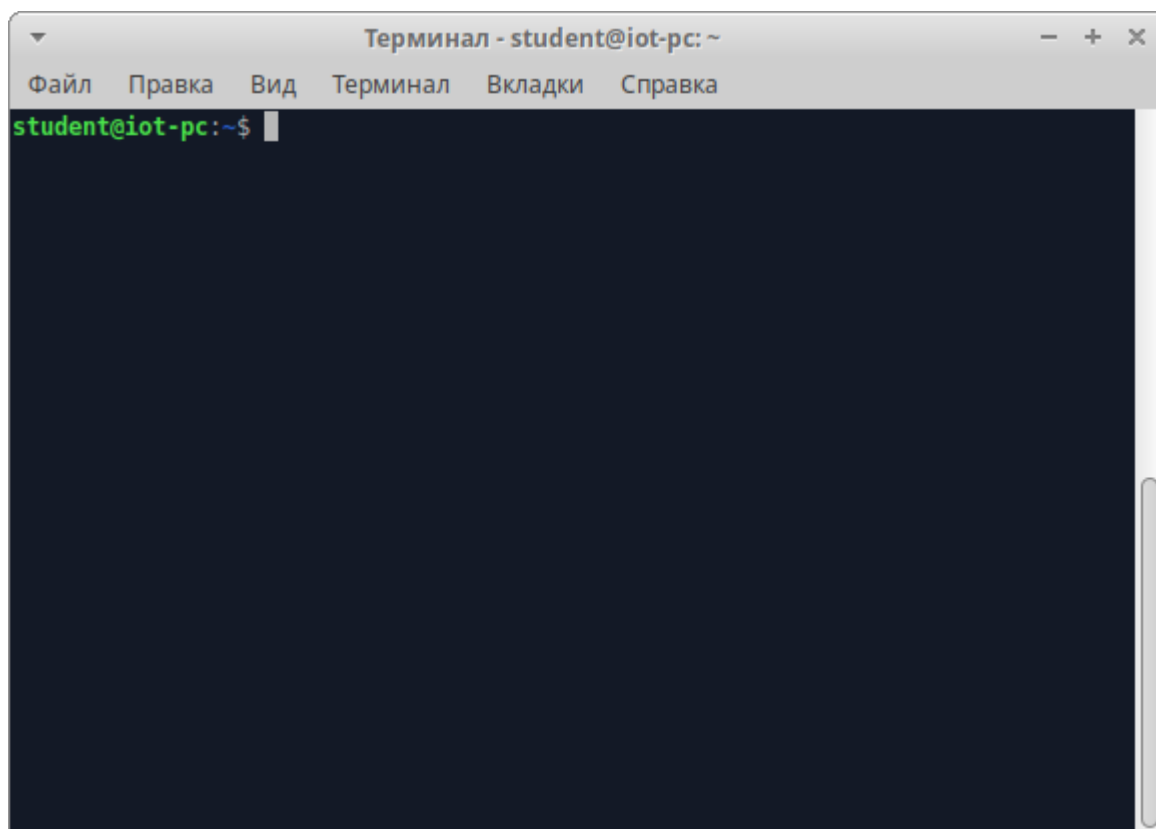
Ревизия 0.2
Новосибирск, 2018

Пользователи Linux, разработчики и администраторы сетей часто пользуются текстовым **интерфейсом командной строки** (он же «Консоль»). Иногда консоль позволяет выполнять задачи в разы быстрее, чем с использованием графического интерфейса. Однако, командная строка совершенно недружелюбна к начинающему пользователю.

Этот мануал призван дать студентам возможность быстрого старта в применении консоли для навигации по операционной системе, редактированию и запуску файлов, а также познакомить с некоторыми ключевыми особенностями и функциями перед началом освоения практического курса IoT школы, не тратя время на поиск информации в разных источниках и не перегружаясь неактуальными на данном этапе знаниями.

Этот мануал разрабатывался за работой в 32-разрядной операционной системе Xubuntu 18.04.1, которая была запущена на виртуальной машине. Был создан один пользователь «student», и в его домашнюю директорию была помещена некая структура из папок и файлов, на примере которой будет происходить знакомство с командами.

Когда вы открываете терминал, то попадаете в домашнюю папку текущего пользователя. Вот как это выглядит:



Строка приглашения ко вводу команд начинается с имени пользователя и имени машины, разделённые символом @. Это очень удобно, потому что у вас одновременно может быть открыто несколько терминалов от имени разных пользователей одной машины, или терминал вообще может быть связан с удалённой машиной по SSH, и, не будь этих подсказок, можно легко было бы запутаться в том, кто где и что делает.

На приведённом скриншоте видно, что терминал работает от имени пользователя «student», а машина называется «iot-pc». Знак тильды (~) – это обозначение домашней директории текущего пользователя. Когда текущая директория сменится на другую, то вместо тильды будет показываться соответствующий путь.

Давайте введём нашу первую команду «ls». Эта команда показывает содержимое текущей директории.

В нашем примере видно, что в домашней директории располагаются папки «data» и «Рабочий стол», а также скрипт на питоне «helloscript.py» и два текстовых файла «mytext.txt» и «list.txt».

```
student@iot-pc:~$ ls
data  helloscript.py  list.txt  mytext.txt  'Рабочий стол'
student@iot-pc:~$
```

Команды консоли являются регистрозависимыми, это означает, что, например, «LS» или «Ls» будут делать не то же самое, что «ls».

У команд могут быть так называемые «ключи» – параметры запуска. Ключи бывают длинные и короткие. Примеры длинных ключей: «--recursive», «--no-target-directory», примеры коротких ключей: «-r», «-f». Как видите, длинные ключи обычно начинаются с «--», а короткие – с «-». Кроме этого, короткие ключи можно объединять в один, например, вместо «-x -v -z» можно записать «-xvz». Список **всех ключей** команды можно посмотреть, запустив эту команду с ключом «--help». Ниже пример вывода списка ключей для команды «ls». Это лишь фрагмент, поскольку ключей очень много, и все не влезли на скриншот.

```
student@iot-pc:~$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILES (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
-a, --all                        do not ignore entries starting with .
-A, --almost-all               do not list implied . and ..
--author                        with -l, print the author of each file
-b, --escape                    print C-style escapes for nongraphic characters
--block-size=SIZE              scale sizes by SIZE before printing them; e.g.,
                               '--block-size=M' prints sizes in units of
                               1,048,576 bytes; see SIZE format below
-B, --ignore-backups            do not list implied entries ending with ~
-c                              with -lt: sort by, and show, ctime (time of last
                               modification of file status information);
                               with -l: show ctime and sort by name;
                               otherwise: sort by ctime, newest first
-C                              list entries by columns
--color[=WHEN]                 colorize the output; WHEN can be 'always' (default
                               if omitted), 'auto', or 'never'; more info below
-d, --directory                list directories themselves, not their contents
-D, --dired                     generate output designed for Emacs' dired mode
-f                              do not sort, enable -aU, disable -ls --color
-F, --classify                 append indicator (one of */=>@|) to entries
--file-type                     likewise, except do not append '*'
--format=WORD                   across -x, commas -m, horizontal -x, long -l,
                               single-column -l, verbose -l, vertical -C
--full-time                     like -l --time-style=full-iso
-g                              like -l, but do not list owner
--group-directories-first      group directories before files;
                               can be augmented with a --sort option, but any
                               use of --sort=none (-U) disables grouping
-G, --no-group                  in a long listing, don't print group names
-h, --human-readable            with -l and/or -s, print human readable sizes
                               (e.g., 1K 234M 2G)
--si                           likewise, but use powers of 1000 not 1024
-H, --dereference-command-line
```

Также здесь можно заметить, что некоторые короткие и длинные ключи дублируют друг друга, например «-B» и «--ignore-backups». Наличие двух форм записи одного и того же ключа имеет смысл, так как зависимости от ситуации вам может понадобиться либо быстрый способ, либо человекочитаемый.

Когда текста в терминале много – как сейчас – для его прокрутки вверх и вниз можно использовать комбинации клавиш «Shift + PgUp» и «Shift + PgDn», соответственно.

Давайте запустим команду «ls» с ключом «-l», который означает вывод в виде списка, и вот что получится:

```
student@iot-pc:~$ ls -l
total 20
drwxrwxr-x 2 student student 4096 сен 10 11:35 data
-rw-rw-r-- 1 student student 40 сен 10 11:24 helloworld.py
-rw-rw-r-- 1 student student 659 сен 10 12:19 list.txt
-rw-rw-r-- 1 student student 60 сен 10 11:23 mytext.txt
drwxrwxr-x 2 student student 4096 сен 10 11:36 'Рабочий стол'
student@iot-pc:~$
```

Каждый файл и каждая директория оказались в отдельной строке, и отобразилась дополнительная информация: дата и время последнего изменения, размер, владелец и права. О правах мы поговорим чуть позже, поскольку эта тема достойна отдельного внимания.

Команда «ls» может выводить содержимое не только текущей директории, но и вообще любой, нужно лишь только её указать. Например, в нашей домашней директории есть папка «data», и чтобы вывести её содержимое нужно написать команду «ls -l data».

Здесь нужно ввести в терминале «ls -l» снова, а мы это делали только что, и резонно в этом месте задать вопрос: «нельзя ли как-то бесплатно повторить ввод предыдущей команды?» Ответ — МОЖНО! В консоли предусмотрена **функция возврата к предыдущей команде** с помощью клавиши «↑» (стрелка вверх). И это одна из самых крутых функций консоли, её использование очень упрощает жизнь Linux-разработчика. Команды можно также листать в обратную сторону с помощью клавиши «↓» (стрелка вниз).

Итак, остаётся только дописать к нашей предыдущей команде «data», и всё готово:

```
student@iot-pc:~$ ls -l data
total 12
-rw-rw-r-- 1 student student 245 сен 10 15:03 random-bytes
-rw-rw-r-- 1 student student 110 сен 10 11:35 user1
-rw-rw-r-- 1 student student 111 сен 10 11:35 user2
student@iot-pc:~$
```

Мы находим три файла: «user1», «user2» и «random-bytes», которые не имеют расширения. В линуксе такое часто бывает — файл вообще не обязан иметь расширение. В этих файлах содержатся тексты, но посмотрим мы их позже.

Для смены текущей директории применяется команда «cd». Эта команда без параметров вернёт вас домой из любой директории. Есть два способа навигации командой «cd»: через абсолютный путь или через относительный путь.

Относительный путь — это когда „отсчёт“ ведётся от текущей директории. Например, команда «cd data» приведёт в подпапку «data», а «cd ../» приведёт в папку уровнем выше:

```
student@iot-pc:~$ cd ../
student@iot-pc:/home$
```

Теперь мы находимся в папке «/home». Если подняться ещё на уровень выше, то мы окажемся в корне:

```
student@iot-pc:/home$ cd ../
student@iot-pc:/$
```

Корень («/») – это место, с которого начинаются все пути в Linux. Здесь находятся все системные вещи:

```
student@iot-pc:/$ ls
bin      dev      initrd.img      lost+found  opt      run      srv      tmp      vmlinuz
boot     etc      initrd.img.old  media      proc     sbin     swapfile  usr
cdrom    home    lib            mnt        root     snap     sys       var
student@iot-pc:/$
```

Теперь перейдём в нашу папку «data» через абсолютный путь, который всегда начинается с корня, с помощью команды «**cd /home/student/data**».

```
student@iot-pc:/$ cd /home/student/data
student@iot-pc:~/data$
```

Как вы помните, в этой папке лежат три файла без расширений. Содержимое файла можно посмотреть командой «**cat**»:

```
student@iot-pc:~/data$ cat user1
192 242 3 88187230498
155 879 4 09804395853
992 475 3 01928301932
285 195 8 65418198123
596 524 1 95642154231
student@iot-pc:~/data$
```

Ещё одна очень крутая функция консоли – это **автодополнение**. Допустим, мы хотим вывести содержимое файла «random-bytes» командой «**cat**». Мы набираем «**cat**», затем пишем один символ «**r**», после чего нажимаем клавишу «**Tab**», и оставшиеся символы «andom-bytes» подставляются автоматически. Эта функция тоже значительно ускоряет работу в консоли и просто необходима разработчику. Автодополнение работает не только для названий файлов, но и для команд консоли.

Поскольку у нас два файла, у которых название начинается одинаково: «user1» и «user2», то если мы напишем «**cat u**» и нажмем «**Tab**», автодополнение подставит максимально длинную одинаковую часть двух названий – «ser», после чего пригласит нас ввести уточнение. Повторным нажатием «**Tab**» можно вывести список всех файлов, название которых начинается на «user»:

```
student@iot-pc:~/data$ cat user
user1 user2
student@iot-pc:~/data$ cat user
```

Команда «**find**» позволяет найти путь, по которому лежит файл. Давайте поищем файл «random-bytes» в своей домашней директории. После команды «**find**» следует указать, где производится поиск, например, в домашней директории, которая обозначается тильдой; а затем после ключа «**-name**» указывается имя искомого файла:

```
student@iot-pc:~$ find ~ -name "random*"
/home/student/data/random-bytes
student@iot-pc:~$
```

Команда возвращает абсолютный путь до файла.

Команда «**mkdir**» создаёт папку. Необходимо указать название создаваемой папки:

```
student@iot-pc:~/data$ mkdir source
student@iot-pc:~/data$ ls -l
total 16
-rw-rw-r-- 1 student student 245 сен 10 15:03 random-bytes
drwxrwxr-x 2 student student 4096 сен 10 15:14 source
-rw-rw-r-- 1 student student 110 сен 10 11:35 user1
-rw-rw-r-- 1 student student 111 сен 10 11:35 user2
student@iot-pc:~/data$
```

Видно, что мы создали папку «*source*» внутри «*data*», и она там появилась.

Файл можно скопировать командой «**cp**», при этом необходимо указать что и куда копируется. Например, скопируем файл «*random-bytes*» в папку «*source*»:

```
student@iot-pc:~/data$ cp random-bytes source/
student@iot-pc:~/data$ ls source/
random-bytes
student@iot-pc:~/data$
```

Обратите внимание, что в написании команды после «*source*» идет «/», что означает именно складывание в папку. Если не поставить слэш, то интерпретатор команд подумает, что вы хотите положить файл в этой же папке, но с другим именем. Сменить имя при копировании очень легко, для этого достаточно указать новое имя:

```
student@iot-pc:~/data$ cp random-bytes source/not-so-random-bytes
student@iot-pc:~/data$ ls -l source/
total 8
-rw-rw-r-- 1 student student 245 сен 10 15:25 not-so-random-bytes
-rw-rw-r-- 1 student student 245 сен 10 15:17 random-bytes
student@iot-pc:~/data$
```

Можно скопировать сразу несколько файлов по шаблону. Для создания шаблона используется символ «*****» (звёздочка), который заменяет любой текст. Ниже по шаблону «*user**» одной командой копируются файлы «*user1*» и «*user2*»

```
student@iot-pc:~/data$ cp user* source/
student@iot-pc:~/data$ ls -l source
total 16
-rw-rw-r-- 1 student student 245 сен 10 15:25 not-so-random-bytes
-rw-rw-r-- 1 student student 245 сен 10 15:17 random-bytes
-rw-rw-r-- 1 student student 110 сен 10 15:31 user1
-rw-rw-r-- 1 student student 111 сен 10 15:31 user2
student@iot-pc:~/data$
```

Перемещается файл с помощью команды «**mv**». Эта команда работает аналогично «**cp**» с той лишь разницей, что исходный файл удаляется. В примере ниже показано, как файл «*random-bytes*» переименовывается в «*trash*»:

```

student@iot-pc:~/data/source$ ls -l
total 16
-rw-rw-r-- 1 student student 245 сен 10 15:25 not-so-random-bytes
-rw-rw-r-- 1 student student 245 сен 10 15:17 random-bytes
-rw-rw-r-- 1 student student 110 сен 10 15:31 user1
-rw-rw-r-- 1 student student 111 сен 10 15:31 user2
student@iot-pc:~/data/source$ mv random-bytes trash
student@iot-pc:~/data/source$ ls -l
total 16
-rw-rw-r-- 1 student student 245 сен 10 15:25 not-so-random-bytes
-rw-rw-r-- 1 student student 245 сен 10 15:17 trash
-rw-rw-r-- 1 student student 110 сен 10 15:31 user1
-rw-rw-r-- 1 student student 111 сен 10 15:31 user2
student@iot-pc:~/data/source$

```

Удалить файл можно командой «**rm**» с указанием названия этого файла. В примере ниже мы удаляем файл «trash»:

```

student@iot-pc:~/data/source$ ls -l
total 16
-rw-rw-r-- 1 student student 245 сен 10 15:25 not-so-random-bytes
-rw-rw-r-- 1 student student 245 сен 10 15:17 trash
-rw-rw-r-- 1 student student 110 сен 10 15:31 user1
-rw-rw-r-- 1 student student 111 сен 10 15:31 user2
student@iot-pc:~/data/source$ rm trash
student@iot-pc:~/data/source$ ls -l
total 12
-rw-rw-r-- 1 student student 245 сен 10 15:25 not-so-random-bytes
-rw-rw-r-- 1 student student 110 сен 10 15:31 user1
-rw-rw-r-- 1 student student 111 сен 10 15:31 user2
student@iot-pc:~/data/source$

```

Папку следует удалять вместе с ее содержимым рекурсивно. Для этого используется ключ «**-r**». Как видно из примера ниже, если попытаться удалить папку командой «**rm**» без ключа, то интерпретатор не даёт это сделать, а с ключом всё проходит нормально:

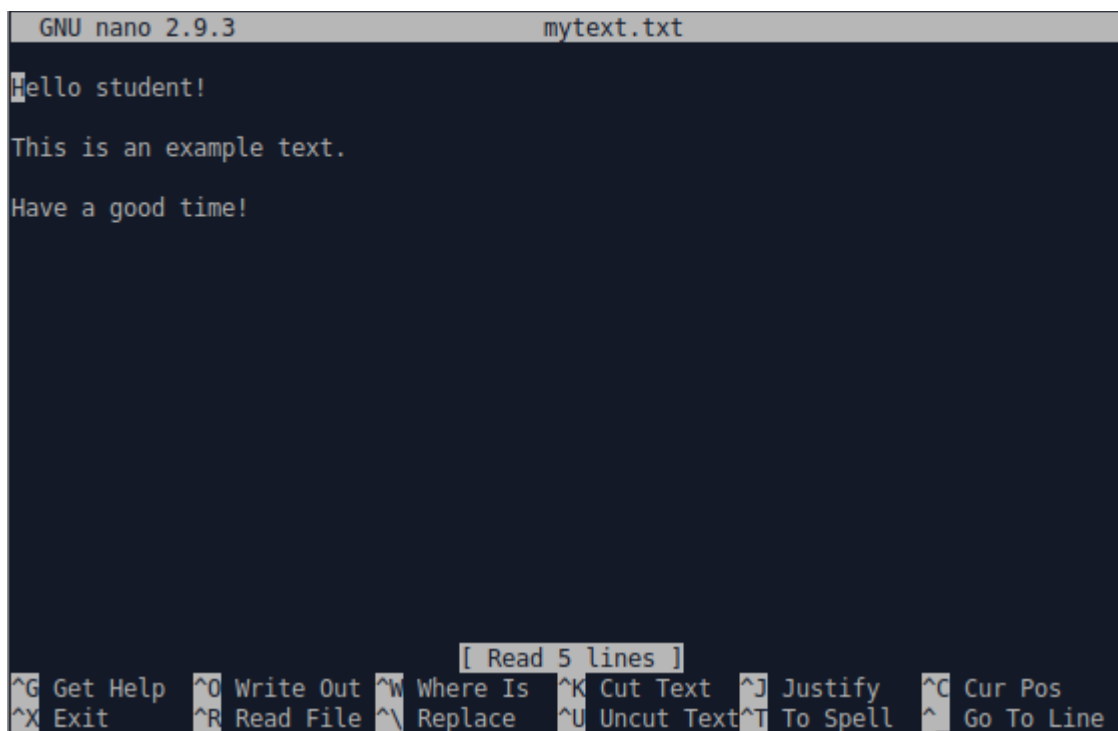
```

student@iot-pc:~/data$ ls -l
total 16
-rw-rw-r-- 1 student student 245 сен 10 15:03 random-bytes
drwxrwxr-x 2 student student 4096 сен 10 15:43 source
-rw-rw-r-- 1 student student 110 сен 10 11:35 user1
-rw-rw-r-- 1 student student 111 сен 10 11:35 user2
student@iot-pc:~/data$ rm source
rm: cannot remove 'source': Is a directory
student@iot-pc:~/data$ rm -r source/
student@iot-pc:~/data$ ls -l
total 12
-rw-rw-r-- 1 student student 245 сен 10 15:03 random-bytes
-rw-rw-r-- 1 student student 110 сен 10 11:35 user1
-rw-rw-r-- 1 student student 111 сен 10 11:35 user2
student@iot-pc:~/data$

```

Для редактирования текста файлов могут использоваться консольные текстовые редакторы, такие как **Vim** или **Emacs**. Их недостаток – в сложности для начинающего пользователя. В то же время, существует очень простой консольный редактор, который называется папо. С помощью папо можно быстро отредактировать небольшой текст или конфигурационный файл.

Для редактирования файла через папо нужно набрать команду «**nano**» с именем этого файла. Если такого файла не существует, то он создастся. Давайте попробуем отредактировать файл «*mytext.txt*» в домашней директории. Для этого нужно набрать «**nano mytext.txt**» (можно пользоваться автодополнением через клавишу «*Tab*», чтобы не *писать* много), и после этого файл откроется в папо:



```
GNU nano 2.9.3 mytext.txt
Hello student!
This is an example text.
Have a good time!

[ Read 5 lines ]
^G Get Help  ^O Write Out ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos
^X Exit      ^R Read File ^_ Replace   ^U Uncut Text ^T To Spell  ^_ Go To Line
```

Навигация по тексту осуществляется стрелками на клавиатуре. Можно вводить или стирать текст, как в любом редакторе. Снизу интерфейса папо находятся подсказки, в них символ «*^*» означает комбинацию с клавишей «*Ctrl*» на клавиатуре. Например, чтобы получить помощь по программе, нужно нажать комбинацию клавиш «*Ctrl + g*».

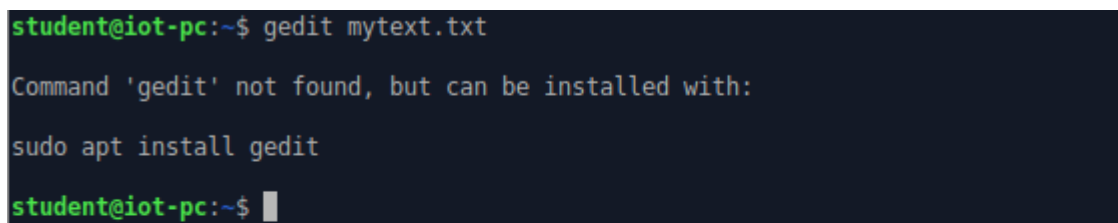
Самые полезные комбинации:

«*Ctrl + w*» – найти текст;

«*Ctrl + o*» – записать изменения;

«*Ctrl + x*» – выйти из папо (если есть незаписанные изменения, то будет задан вопрос о сохранении, на который нужно ответить нажатием клавиши «*y*» или «*n*»).

Если в системе присутствует графический интерфейс, то можно открыть файл в графическом редакторе, например, **gedit**. Попробуем это сделать:



```
student@iot-pc:~$ gedit mytext.txt
Command 'gedit' not found, but can be installed with:
sudo apt install gedit
student@iot-pc:~$
```

Система сообщает, что программа **gedit** не установлена, и подсказывает, что её можно установить командой «**sudo apt install gedit**». Установка любых программ в Linux требует прав **суперпользователя**.

Суперпользователь – это такой пользователь, который может делать с системой всё, что угодно. Обладая такими правами можно даже удалить любые системные файлы, после чего система перестанет работать, поэтому пользоваться ими стоит с осторожностью. Если обычный пользователь имеет разрешение запускать программы от имени суперпользователя,

то он может это делать, используя «sudo» перед командой (пользователи Linux обычно говорят об этом процессе – «запустить что-то от рута»).

Большинство дистрибутивов семейства Linux сопровождаются хранилищем пакетов, которое называется репозиторий или репозитарий (пользователи Linux часто называют его просто «репа»). В нем хранятся предварительно скомпилированные пакеты большинства используемых программ. Устанавливаются пакеты с помощью пакетного менеджера, в Ubuntu и Debian это **apt**, в Fedora и Red Hat это **rpm**, в других дистрибутивах он может быть другой.

Попробуем установить пакет командой «**apt install**»:

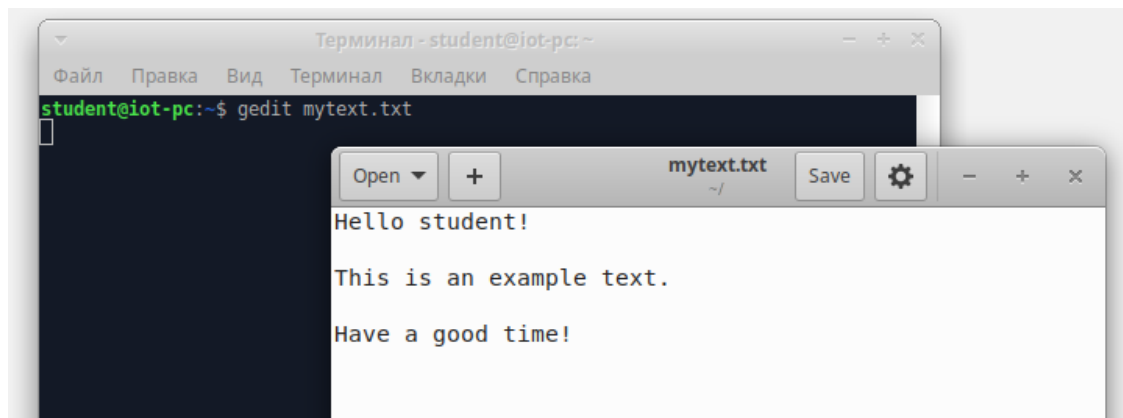
```
student@iot-pc:~$ sudo apt install gedit
[sudo] password for student: █
```

Система спрашивает пароль при повышении привелегий до суперпользователя. При вводе пароля обратите внимание, что не появляются никакие символы (звёздочки, кружочки) – это сделано для того, чтобы подглядывающий за вашим экраном не смог узнать даже сколько символов в вашем пароле.

```
student@iot-pc:~$ sudo apt install gedit
Чтение списков пакетов... Готово
Построение дерева зависимостей
Чтение информации о состоянии... Готово
Будут установлены следующие дополнительные пакеты:
  gedit-common gir1.2-gtksource-3.0 gir1.2-peas-1.0 libpeas-1.0-0
  libpeas-common
Предлагаемые пакеты:
  gedit-plugins
НОВЫЕ пакеты, которые будут установлены:
  gedit gedit-common gir1.2-gtksource-3.0 gir1.2-peas-1.0 libpeas-1.0-0
  libpeas-common
Обновлено 0 пакетов, установлено 6 новых пакетов, для удаления отмечено 0 пакетов,
и 96 пакетов не обновлено.
Необходимо скачать 665 кВ архивов.
После данной операции, объём занятого дискового пространства возрастёт на 5 526
кВ.
Хотите продолжить? [Д/н] █
```

apt показывает, какие у пакета есть зависимости, которые также необходимо будет установить, размер файлов для скачивания и размер файлов после установки. Чтобы продолжить установку достаточно нажать клавишу «Enter». После этого **apt** скачает и установит пакеты, а по завершении освободит командную строку.

Теперь, когда редактор **gedit** установлен, можно открывать файл для редактирования:

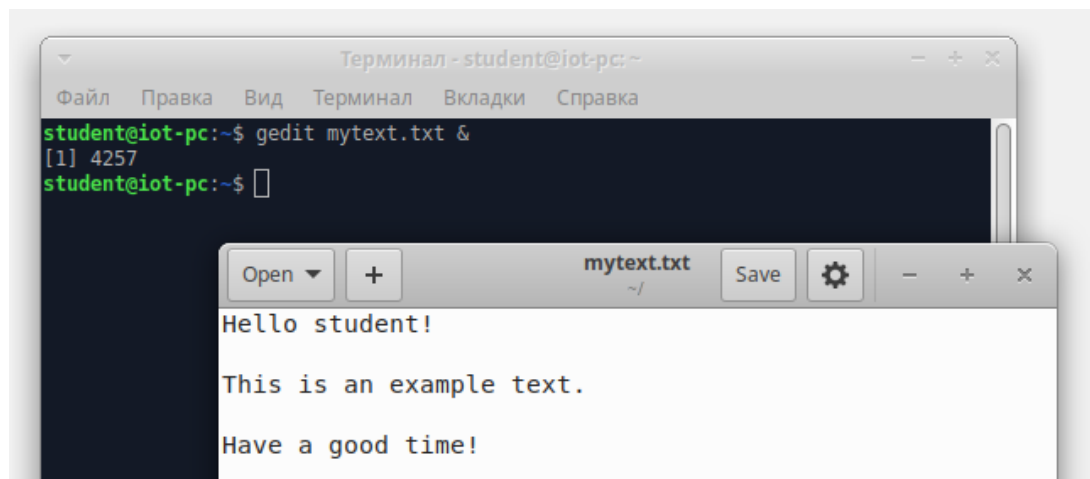


gedit позволяет открывать сразу несколько документов во вкладках и имеет более продвинутые средства редактирования.

В нашем примере есть один важный момент — после запуска **gedit** командой в терминале, терминал остался занят и не позволяет вводить новые команды. В этот терминал теперь будут выводиться сообщения редактора **gedit**. Так будет происходить с любой программой или командой, которая не завершает свою работу мгновенно. Завершить её, находясь в том же терминале, в котором она запущена, можно с помощью комбинации клавиш «*Ctrl + c*».

Как видите, комбинация «*Ctrl + c*» занята под остановку процесса, поэтому использовать её для копирования текста из терминала не получится. Для копирования и вставки текста в терминале применяются комбинации клавиш «*Ctrl + Shift + c*» и «*Ctrl + Shift + v*», соответственно.

Чтобы запустить процесс и при этом сразу освободить терминал, в конце команды следует добавить символ «&». При этом программа запустится в фоновом режиме и система отобразит о номере процесса, который был присвоен программе:



Посмотреть список процессов в системе можно командой «**ps**». Без ключей её вывод не представляет особого интереса. Наиболее полное представление о процессах можно получить, запустив «**ps**» с ключами «**aux**», при этом перед ключами не обязательно добавлять символ «-». то есть просто «**ps aux**». Ниже приведен фрагмент вывода команды:

```
message+ 449 0.0 0.4 7764 4388 ? Ss 10:16 0:03 /usr/bin/dbus-daemon --system --address=systemd: --nofork -
root 480 0.0 1.1 70408 11560 ? Ssl 10:16 0:00 /usr/sbin/NetworkManager --no-daemon
lp 489 0.0 0.3 14456 4056 ? S 10:16 0:00 /usr/lib/cups/notifier/dbus dbus://
root 490 0.0 0.1 11400 1148 ? Ss 10:16 0:00 /sbin/wpa_supplicant -u -s -O /run/wpa_supplicant
avahi 491 0.0 0.2 6160 2656 ? Ss 10:16 0:00 avahi-daemon: running [iot-pc.local]
root 493 0.0 0.6 869600 6820 ? Ssl 10:16 0:00 /usr/lib/snapd/snapd
root 503 0.0 0.9 45560 9840 ? Ssl 10:16 0:00 /usr/lib/policykit-1/polkitd --no-debug
lp 504 0.0 0.3 14456 4088 ? S 10:16 0:00 /usr/lib/cups/notifier/dbus dbus://
lp 505 0.0 0.4 14456 4188 ? S 10:16 0:00 /usr/lib/cups/notifier/dbus dbus://
avahi 536 0.0 0.0 6000 32 ? S 10:16 0:00 avahi-daemon: chroot helper
root 538 0.0 0.8 44208 8404 ? Ssl 10:16 0:00 /usr/sbin/cups-browsed
root 563 0.0 0.1 32228 1940 ? Sl 10:16 0:07 /usr/sbin/VBoxService --pidfile /var/run/vboxadd-service.sh
root 588 0.0 0.8 40220 8296 ? Ssl 10:16 0:00 /usr/sbin/lightdm
whoopsie 606 0.0 0.9 60420 9960 ? Ssl 10:16 0:00 /usr/bin/whoopsie -f
kernoops 615 0.0 0.0 10228 332 ? Ss 10:16 0:00 /usr/sbin/kerneloops --test
kernoops 617 0.0 0.0 10228 328 ? Ss 10:16 0:00 /usr/sbin/kerneloops
root 642 1.3 13.0 280968 133236 tty7 Ssl+ 10:16 4:03 /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth /var/run/ligh
root 644 0.0 0.1 4984 1664 tty1 Ss+ 10:16 0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
root 687 0.0 0.5 29660 6016 ? Sl 10:16 0:00 lightdm --session-child 12 15
student 691 0.0 0.4 12460 4512 ? Ss 10:16 0:00 /lib/systemd/systemd --user
student 692 0.0 0.1 14452 1072 ? S 10:16 0:00 (sd-pam)
student 703 0.0 0.1 2492 1472 ? Ss 10:16 0:00 /bin/sh /etc/xdg/xfce4/xinitrc -- /etc/X11/xinit/xserverrrc
student 716 0.0 0.4 7504 4284 ? Ss 10:16 0:05 /usr/bin/dbus-daemon --session --address=systemd: --nofork
student 792 0.0 0.0 18360 272 ? S 10:16 0:00 /usr/bin/VBoxClient --clipboard
student 793 0.0 0.3 20060 3452 ? Sl 10:16 0:00 /usr/bin/VBoxClient --clipboard
student 802 0.0 0.0 18360 268 ? S 10:16 0:00 /usr/bin/VBoxClient --display
student 803 0.0 0.2 18360 2872 ? S 10:16 0:00 /usr/bin/VBoxClient --display
student 809 0.0 0.0 18360 272 ? S 10:16 0:00 /usr/bin/VBoxClient --seamless
student 810 0.0 0.1 20044 1828 ? Sl 10:16 0:00 /usr/bin/VBoxClient --seamless
student 815 0.0 0.0 18360 268 ? S 10:16 0:00 /usr/bin/VBoxClient --draganddrop
student 816 0.4 0.1 21584 1844 ? Sl 10:16 1:15 /usr/bin/VBoxClient --draganddrop
student 824 0.0 0.0 4804 208 ? Ss 10:16 0:00 /usr/bin/ssh-agent /usr/bin/im-launch startxfce4
student 842 0.0 1.2 48548 12852 ? Sl 10:16 0:00 xfce4-session
student 846 0.0 0.4 10496 4884 ? S 10:16 0:00 /usr/lib/i386-linux-gnu/xfce4/xfconf/xfconfd
student 850 0.0 0.5 42356 5368 ? Sl 10:16 0:00 gnome-keyring-daemon --start
student 854 0.1 2.7 90184 28488 ? Sl 10:16 0:20 xfwm4 --replace
student 859 0.0 1.8 42468 18856 ? Sl 10:16 0:05 xfce4-panel
student 861 0.0 2.6 76688 26892 ? Sl 10:16 0:00 Thunar --daemon
student 863 0.0 3.2 112424 32940 ? Sl 10:16 0:05 xfdesktop
student 868 0.0 3.4 100704 35252 ? Ssl 10:16 0:01 xfsettingsd
```

Здесь содержится информация о пользователе, от имени которого запущен процесс, номер процесса, проценты используемых ресурсов и другие данные.

Чаще всего список процессов вам понадобится, чтобы узнать номер конкретного процесса для его принудительного завершения. Например, вы разрабатываете программу-сервер, запустили текущую её версию, и она заняла какой-то сокет и работает в фоне. Затем вы изменили исходный код программы и хотите запустить её снова. У вас, скорее всего, не получится ничего хорошего, потому что сокет занят старой версией программы. Вам нужно завершить работу старой версии программы, прежде, чем стартовать новую.

Искать самостоятельно в выводе утилиты **ps** некий процесс может оказаться утомительно, и здесь на помощь приходят средства **конвейеризации** для командной строки. Суть конвейера в том, что вывод одной команды можно сразу перенаправить на вход другой команды с помощью символа «|» (вертикальная черта). Например, вывод команды «**ps**» можно перенаправить на вход команды «**grep**», которая выполняет поиск текста:

```
student@iot-pc:~$ ps aux | grep gedit
student 4257 2.0 3.8 120488 39068 pts/0 Sl 15:35 0:01 gedit mytext.txt
student 4291 0.0 0.0 6520 836 pts/0 S+ 15:36 0:00 grep --color=auto gedit
student@iot-pc:~$
```

Так мы нашли наш процесс **gedit**, который до сих пор запущен, и узнали его номер: 4257. Чтобы завершить процесс можно воспользоваться командой «**kill -9**» с указанием номера этого процесса. В нашем случае – «**kill -9 4257**». Программа **gedit** наконец-то закроется. Здесь «-9» выглядит как странный ключ, но на самом деле это не ключ, а сигнал «минус девять», который означает аварийное завершение работы процесса.

Использование конвейера в командной строке не ограничивается «грепанием» (как часто говорят пользователи Linux о поиске с помощью команды «**grep**»), это мощный инструмент, который позволяет выполнять некоторые задачи посредством только лишь командной строки.

В нашей домашней директории лежит скрипт на python, который называется «**helloscript.py**», который печатает в консоль приветственное сообщение. Чтобы его запустить через запуск интерпретатора в консоли, можно воспользоваться командой «**python helloscript.py**»:

```
student@iot-pc:~$ python helloscript.py
Hello world!
student@iot-pc:~$
```

Также вызов интерпретатора можно добавить в сам скрипт, тогда этот скрипт можно будет запускать просто по его названию. Для этого в начало файла нужно добавить строку «**#!/usr/bin/env python**». Вот как будет выглядеть скрипт:

```
student@iot-pc:~$ cat helloscript.py
#!/usr/bin/env python

print "Hello world!"
student@iot-pc:~$
```

Для запуска любого исполняемого файла из папки, в которой он находится, перед именем этого файла нужно добавить «**./**» (точка-слэш). Попробуем запустить наш скрипт:

```
student@iot-pc:~$ ./helloscript.py
bash: ./helloscript.py: Permission denied
student@iot-pc:~$
```

Но он не запускается, потому что не хватает прав. Давайте посмотрим, какие у него права. Для этого воспользуемся командой «**ls -l**», которую мы изучали самой первой:

```
student@iot-pc:~$ ls -l
total 20
drwxrwxr-x 2 student student 4096 сен 12 15:36 data
-rw-rw-r-- 1 student student  44 сен 13 08:19 helloscript.py
-rw-rw-r-- 1 student student  719 сен 12 14:50 list.txt
-rw-rw-r-- 1 student student   60 сен 12 14:42 mytext.txt
drwxrwxr-x 2 student student 4096 сен 12 16:25 'Рабочий стол'
student@iot-pc:~$
```

В самом левом столбце перед названием файла или папки обозначены права. Для файла «**helloscript.py**» права выглядят так: «**-rw-rw-r--**». Самый первый символ обозначает, является ли элемент списка папкой. Если это папка, то на этом месте стоит «**d**», если нет, то «**-**». Далее идут девять символов, которые делятся на три части по три. Первая – это права владельца файла на чтение, запись и исполнение. Для нашего файла «**helloscript.py**» видно, что владелец может его читать («**r**»), может в него писать («**w**») и не может его исполнять («**-**»). Вторая часть показывает права всех пользователей, которые входят в ту же группу, что и владелец (то есть в группу «**student**»). У них права на чтение, запись и исполнение такие же – «**rw**». Третья часть показывает права всех других пользователей. Видно, что они могут только читать файл и не могут в него писать и его исполнять («**-r--**»).

Владелец файла или суперпользователь могут изменить права на файл с помощью команды «**chmod**». Например, чтобы дать всем пользователям право на исполнение файла, нужно воспользоваться командой «**chmod +x**»:

```
student@iot-pc:~$ ls -l
total 20
drwxrwxr-x 2 student student 4096 сен 12 15:36 data
-rw-rw-r-- 1 student student 44 сен 13 08:19 helloworld.py
-rw-rw-r-- 1 student student 719 сен 12 14:50 list.txt
-rw-rw-r-- 1 student student 60 сен 12 14:42 mytext.txt
drwxrwxr-x 2 student student 4096 сен 12 16:25 'Рабочий стол'
student@iot-pc:~$ chmod +x helloworld.py
student@iot-pc:~$ ls -l
total 20
drwxrwxr-x 2 student student 4096 сен 12 15:36 data
-rwxrwxr-x 1 student student 44 сен 13 08:19 helloworld.py
-rw-rw-r-- 1 student student 719 сен 12 14:50 list.txt
-rw-rw-r-- 1 student student 60 сен 12 14:42 mytext.txt
drwxrwxr-x 2 student student 4096 сен 12 16:25 'Рабочий стол'
student@iot-pc:~$
```

Видно, что у «**helloworld.py**» права стали «**-rwxrwxr-x**». Это означает, что теперь скрипт стал исполняемым, и его могут запускать все пользователи (и, кстати, его имя в терминале окрасилось в зелёный цвет). Теперь его можно запустить через «./»:

```
student@iot-pc:~$ ./helloworld.py
Hello world!
student@iot-pc:~$
```

Теперь давайте напишем простую программу на языке C, скомпилируем её и запустим. Создадим папку «**cprog**», перейдем в неё и создадим там файл «**hello.c**»:

```
student@iot-pc:~$ mkdir cprog
student@iot-pc:~$ cd cprog
student@iot-pc:~/cprog$ nano hello.c
```

Теперь напишем текст программы:

```
GNU nano 2.9.3 helloworld.c

#include <stdio.h>

int main(){
    printf("Hello from C app\r\n");
    return 0;
}
```

Для компиляции программ на C используется компилятор **GCC (GNU Compiler Collection)**. Этот компилятор является очень распространенным и поддерживает очень много разных платформ.

Команде «**gcc**» на вход нужно подать файл с исходным кодом и можно указать имя выходного файла через ключ «**-o**». Давайте скомпилируем программу, посмотрим, что появился исполняемый файл («**бинарник**») и запустим его:

```
student@iot-pc:~/cprog$ gcc hello.c -o hello
student@iot-pc:~/cprog$ ls
hello hello.c
student@iot-pc:~/cprog$ ./hello
Hello from C app
student@iot-pc:~/cprog$
```

Если в вашем проекте на языке C больше одного файла с исходным кодом, то имеет смысл обратить своё внимание на **make** – утилиту для автоматизации сборки. Программа **make** выполняет команды согласно правилам, указанным в файле, который называется **Makefile** и имеет строгий синтаксис.

Для начала создадим три файла – «**hello.c**», «**hellofunc.c**» и «**hellofunc.h**» со следующим содержанием:

```
GNU nano 2.9.3 hello.c Modified
#include <hellofunc.h>

int main(){
    hellofunc();
    return 0;
}
```

```
GNU nano 2.9.3 hellofunc.c Modified
#include <stdio.h>
#include <hellofunc.h>

void hellofunc(void){
    printf("Hello from MAKEd app\r\n");
}
```

```
GNU nano 2.9.3 hellofunc.h
void hellofunc(void);
```

После того, как файлы созданы, создадим **Makefile**:

```
GNU nano 2.9.3 Makefile
CC=gcc
CFLAGS=-I.
DEPS = hellofunc.h

hellomake: hello.o hellofunc.o
    $(CC) -o hellomake hello.o hellofunc.o
```

Здесь мы определили три константы: «**CC**» – название компилятора, «**CFLAGS**» – флаги компилятора, «**DEPS**» – файлы зависимостей. Затем мы создали цель «**hellomake**» и после двоеточия указали для неё зависимости в виде двух объектных файлов «**hello.o**» и «**hellofunc.o**». В следующей строке приводится команда, которую должен выполнить **make** для достижения цели. Видно, что он должен собрать файл «**hellomake**» компилятором, указанным в константе «**CC**», из объектных файлов «**hello.o**» и «**hellofunc.o**». Важно, что строка должна начинаться с табуляции.

После сохранения мейкфайла можно запустить команду «**make**» и запустить созданный бинарник:

```
student@iot-pc:~/cprog$ make
gcc -I. -c -o hello.o hello.c
gcc -I. -c -o hellofunc.o hellofunc.c
gcc -o hellomake hello.o hellofunc.o
student@iot-pc:~/cprog$ ./hellomake
Hello from MAKEd app
student@iot-pc:~/cprog$
```

Утилита «**make**» часто используется для сборки программ под Linux и прошивок микроконтроллеров и у неё огромные возможности по тонкой настройке, которые стоит изучить самостоятельно.

А теперь сюрприз! Всё, что мы до этого вводили в терминал, можно посмотреть командой «**history**»:

```
student@iot-pc:~/data$ history
1 history
2 clear
3 uname -a
4 ls
5 ls --help
6 ls -l
7 ls -l data
8 cd ../
9 cd /home/student/data
10 cat user1
11 find ~ -name "random"
```

Конечно же, в истории можно выполнить поиск, чтобы быстро найти ранее введенную команду. Для этого необходимо нажать комбинацию клавиш «**Ctrl + r**», которая включает режим «*reverse-i-search*», после чего ввести сочетание символов, которое присутствовало в искомой команде. Повторное нажатие «**Ctrl + r**» позволяет переключать по очереди все команды, в которых встретится искомое сочетание:

```
(reverse-i-search)`ls': ls -l data
```

На этом мануал завершается.

Желаю всем успехов в обучении!

Связаться со мной можно по электронной почте nes90@mail.ru