

Введение в UNIX-системы

Работа в графическом и консольном режиме

Консольный и графический режим в Linux. Консольные команды. Создание и редактирование файлов.

Оглавление

[Введение](#)

[Работа в консоли](#)

[Переключение между виртуальными терминалами](#)

[Работа с каталогами в оболочке](#)

[Практическая задача по перемещению файлов](#)

[Вспомогательные клавиши для эффективной работы в командной строке](#)

[Редакторы и создание файлов](#)

[Простейшие способы создания файлов](#)

[Создание и редактирование файлов](#)

[mcedit](#)

[nano](#)

[vi](#)

[Как найти слово в vi](#)

[Основные элементы Linux](#)

[UNIX way](#)

[Ядро](#)

[Системные утилиты, оболочка](#)

[Демоны](#)

[Приложения пользователя, графический интерфейс X Window System](#)

[X-Window и оконные менеджеры](#)

[Основные элементы рабочего стола](#)

[Пример запуска терминала](#)

[Рабочие столы](#)

[Использование рабочих мест](#)

[Выключение системы](#)

[Процессы, потоки, потоки ядра](#)

[Утилита ps \(Process Show\)](#)

[Утилиты top и htop](#)

[Системный монитор в X-Window](#)

[Порядок запуска процесса](#)

[Утилита kill и сигналы процессам](#)

[Перенаправление потоков ввода-вывода](#)

[Установка программ в Ubuntu](#)

[Ключевые понятия](#)

[Основной способ установки программ. Использование официальных репозиториев](#)

[Использование сторонних репозиториев](#)

[Непосредственная установка из deb-пакетов](#)

[Установка программ с собственным инсталлятором](#)

[Установка программ, которые не нуждаются в инсталляции](#)

[Установка программ из исходных текстов](#)

[Репозитории в UBUNTU](#)

[Структура репозитория Ubuntu](#)

[Подключение репозитория](#)

[Подключение при помощи графического интерфейса](#)

[При помощи консоли \(рекомендуемый способ\)](#)

[Устройство репозитория](#)

[Защита репозитория](#)

[Логи, просмотр логов](#)

[Встроенная документация. Команды man, apropos](#)

[Архиваторы и компрессоры в Linux](#)

[Утилита tar \(Tape ARchiver\)](#)

[Создание архивов с tar](#)

[Просмотр оглавления архива](#)

[Извлечь файлы из архива](#)

[Компрессоры](#)

[gzip](#)

[gunzip](#)

[zcat](#)

[Практическое задание](#)

[Дополнительные материалы](#)

[Используемая литература](#)

Введение

Linux можно использовать как в десктопной версии, так и в серверной. В десктопной применяется X11-Server, и вы можете работать в окружении рабочего стола, схожего с знакомым нам по Windows. Таким образом, можно установить Linux на старый компьютер и использовать его. Часто Linux нужен, чтобы сэкономить на лицензии и оборудовать класс информатики или офис. В Linux много графических программ, при этом не все программы, существующие в Windows, могут быть доступны для Linux. В этом случае используют Wine либо Virtualbox, или VMWare с образом Windows для запуска приложений. Для организации офисных рабочих мест могут применяться сервер приложений и доступ к Windows-приложениям с RDP.

При этом основное достоинство Linux — применение в серверном администрировании. В этом случае используется консоль. Если у вас есть непосредственный доступ к компьютеру, переключение между терминалами (которые называются TTY1 — TTY6) происходит по нажатию клавиш Alt-F1... Alt-F6. Нажатие Alt-F1 переводит в режим X11. Он запускается командой **startx**. Если вы работаете в графическом режиме, можете запустить эмулятор терминала с помощью команды Ctrl-Alt-T либо перейти в настоящий терминал с помощью сочетаний Ctrl-Alt-F2... Ctrl-Alt-F6. Также возможна удаленная работа. При SSH-подключении на каждое подключение будет только один виртуальный терминал. При работе с помощью VNC есть доступ ко всем TTY1-TTY6 и графическому режиму. При SSH тоже можно запускать графические приложения, если установлен X11-сервер (в Linux или macOS по умолчанию, либо дополнительно, например Xming или MobaXterm для Windows).

Работа в консоли

Переключение между виртуальными терминалами

Помимо графического пользовательского интерфейса, в любой Linux-системе есть возможность работы с текстовым терминалом. Терминалы использовались с самых ранних версий UNIX и представляли собой простейшие символьные устройства ввода-вывода. В самых первых реализациях это устройство выглядело как печатная машинка, совмещенная с принтером.



Телетайп ARS-33, используемый как терминал

Автор: Dominic Alves from Brighton, England — ASR 33 Teletype, CC BY 2.0.

<https://commons.wikimedia.org/w/index.php?curid=4259023>

Терминал позволял вводить текстовые команды и печатать ответ системы в текстовом виде. Секрет популярности такого интерфейса был в простоте физической реализации устройства (в простейшем варианте достаточно всего трех проводов последовательного интерфейса) и программной поддержке, которая не требовала драйверов клавиатуры и видеокарты. Постепенно устройства становились более продвинутыми, позволяли использовать функциональные клавиши и управляющие последовательности для выделения текста цветом. Но их суть оставалась неизменной: все тот же последовательный текстовый ввод-вывод. Аппаратные терминалы до сих пор используются в RISC UNIX-системах производства IBM, HP и Oracle (ex-Sun). В Linux пользователь сталкивается с виртуальной разновидностью терминалов.

Виртуальные терминалы используются на серверах, где графический интерфейс не установлен. На системах с установленным графическим интерфейсом виртуальные терминалы можно применять параллельно с GUI. Это бывает полезно, когда графический интерфейс оказывается неработоспособным. Чтобы переключиться из графического интерфейса в текстовый терминал, нажмите комбинацию Ctrl+Alt+F2 (либо F3..F6). После переключения вы увидите приглашение зарегистрироваться в системе:

```
Ubuntu 18.04.2 LTS ub18 tty2
ub18 login: _
```

Вместо **ub18** будет имя системы, которое указали при установке, **tty2** — номер виртуального терминала (tty3 ... tty6).

После успешной регистрации в системе выдается приглашение вида `user@host~$`, где **user** — регистрационное имя, **host** — имя системы, заданное при установке, а символ `~` — домашний каталог пользователя. Такая подсказка (или приглашение) означает, что запущен интерпретатор команд и далее можно вводить имена системных утилит и запускать другие команды.

```
Ubuntu 18.04.2 LTS ub18 tty2
ub18 login: geek
Password:
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.18.0-20-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

Могут быть обновлены 299 пакетов.
125 обновлений касаются безопасности системы.

Your Hardware Enablement Stack (HWE) is supported until April 2023.

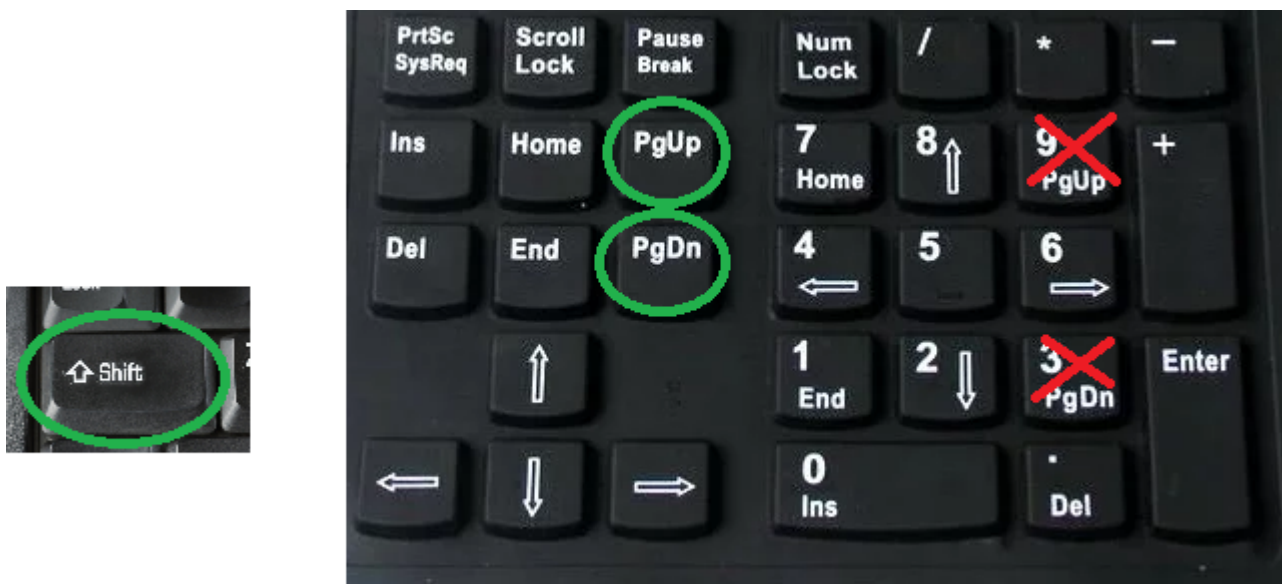
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

geek@ub18:~$
```

По умолчанию в системе есть шесть виртуальных терминалов, переключаться между которыми можно с помощью комбинации клавиш <Alt>+F1 ... <Alt>+F6. При регистрации номер виртуального терминала указан в системном приглашении как **ttyN**, где N = 1..6. Для возврата в графический интерфейс используйте <Alt>+F1.

Еще одна полезная консольная комбинация клавиш — Shift-PgUP / Shift-PgDN. Позволяет прокручивать текст на консоли постранично. Обратите внимание, что работать будут только кнопки PgUp и PgDn, не совмещенные с цифровой клавиатурой. Даже если Num Lock выключен, PgUp/9 и PgDn/3 не заработают.



Это бывает необходимо, например, когда текст ошибки появился на консоли и тут же исчез за границами экрана.

Чтобы запустить терминальную сессию с командным интерпретатором из графического интерфейса, можно использовать приложение «Терминал», которое вы можете запустить с помощью команды клавиш Ctrl-Alt-T. Это эмулятор терминала. Термин «эмулятор» означает, что приложение эмулирует возможности аппаратного терминала определенного типа. Эмулятор поддерживает управляющие последовательности устройства, которые позволяют выводить цветной текст, мигающие символы и переносить курсор. Также эмулируются функциональные клавиши терминала и другие особенности.

Работа с каталогами в оболочке

Мы получили доступ к интерпретатору команд (в нашем случае эта программа называется **bash**). Теперь можно вводить имена системных утилит и запускать программы. Далее приведем пример типичной сессии, в которой требуется отсортировать большое количество файлов, разложив их по разным каталогам. Мы изучим команды перемещения по файловой системе, создания каталогов и перемещения файлов.

У каждого процесса в Unix есть атрибут, который называется «текущий каталог». Есть такой атрибут и у командного интерпретатора. По умолчанию, если не указывать путь к файлам, все команды будут работать с файлами из текущего каталога. Поэтому для начала определим свое местонахождение в файловой системе, или текущий каталог командного интерпретатора. Его показывает команда **pwd**, которую мы вводим после приглашения командной строки [user@user-virtual-machine:~\\$](#). Команда передается на исполнение только после ввода <Enter>, до этого момента ее можно редактировать, перемещать курсор стрелками, а также использовать <Backspace> для корректировки ввода. Затем

команда выводит результат и завершает работу, после чего на новой строке вновь появляется приглашение.

```
user@user-virtual-machine:~$ pwd
/home/user
user@user-virtual-machine:~$
```

В UNIX используется иерархическая файловая система. Самый верхний уровень иерархии — / (корневой каталог). В корневом каталоге есть подкаталог **home**, в котором, в свою очередь, подкаталог **user**, служащий домашним каталогом пользователя **user**, от имени которого выполняется сессия. **/home/user** задает полный путь домашнего каталога, то есть путь, начинающийся с корневого каталога. Еще есть относительный путь, который задает путь относительно текущего каталога. В системной подсказке **user@user-virtual-machine:~\$** используется один из вариантов относительного пути, где символ «тильда» (~) обозначает домашний каталог пользователя.

Изменим текущий каталог, переместившись на уровень выше, в каталог **/home**, с помощью команды **cd** (Change Directory):

```
user@user-virtual-machine:~$ cd ..
user@user-virtual-machine:/home$
```

Параметр команды **cd** — каталог, куда мы желаем попасть, новый текущий каталог. Можно было использовать полный путь, набрав **cd /home**, но мы использовали еще один вариант относительного пути, в котором двоеточие означает вышестоящий в иерархии каталог (родительский). Как видим, текущий каталог приглашения интерпретатора также изменился. Теперь вернемся обратно в домашний каталог, используя относительный путь:

```
user@user-virtual-machine:~$ cd user
user@user-virtual-machine:~$
```

Того же результата мы могли добиться командами **cd /home/user**, **cd ~** или просто **cd** без параметров.

Теперь исследуем домашний каталог с помощью команды **ls** (LiSt):

```
user@user-virtual-machine:~$ ls
examples.desktop  Документы  Изображения  Общедоступные  Шаблоны
Видео            Загрузки  Музыка       Рабочий стол
```

По умолчанию **ls** показывает текущий каталог, но можно задать любой интересующий нас каталог как параметр команды. В примере часть файлов показывается синим цветом: на самом деле это не файлы, а каталоги. Выделение каталогов синим — особенности настройки команды в данной системе. В других Linux / UNIX или в другом терминале все может быть иначе — чтобы наверняка знать, где файл, а где каталог, используйте **ls** с опцией **-l**:

```

user@user-virtual-machine:~$ cd /
user@user-virtual-machine:/$ ls -l
total 100
drwxr-xr-x  2 root root  4096 дек 15 22:33 bin
drwxr-xr-x  3 root root  4096 дек 15 22:41 boot
drwxrwxr-x  2 root root  4096 дек  9 15:33 cdrom
drwxr-xr-x 19 root root 4320 дек 15 21:55 dev
drwxr-xr-x 134 root root 12288 дек 16 16:21 etc
drwxr-xr-x  3 root root  4096 дек  9 15:35 home
lrwxrwxrwx  1 root root   33 дек 15 22:39 initrd.img -> boot/initrd.img-4.10.0-42-generic
lrwxrwxrwx  1 root root   32 дек  9 15:38 initrd.img.old -> boot/initrd.img-4.8.0-36-generic
drwxr-xr-x 22 root root  4096 дек  9 15:41 lib
drwxr-xr-x  2 root root  4096 дек 15 22:31 lib64
drwx----- 2 root root 16384 дек  9 15:27 lost+found
drwxr-xr-x  2 root root  4096 фев 15 2017 media
drwxr-xr-x  2 root root  4096 фев 15 2017 mnt
drwxr-xr-x  2 root root  4096 фев 15 2017 opt
dr-xr-xr-x 209 root root    0 дек 15 21:54 proc
drwx----- 6 root root  4096 дек 16 16:06 root
drwxr-xr-x 29 root root  1040 дек 16 17:04 run
drwxr-xr-x  2 root root 12288 дек 15 22:34 sbin
drwxr-xr-x  2 root root  4096 янв 14 2017 snap
drwxr-xr-x  2 root root  4096 фев 15 2017 srv
dr-xr-xr-x 13 root root    0 дек 16 17:05 sys
drwxrwxrwt 10 root root  4096 дек 16 17:20 tmp
drwxr-xr-x 11 root root  4096 фев 15 2017 usr
drwxr-xr-x 15 root root  4096 дек 16 15:56 var
lrwxrwxrwx  1 root root   30 дек 15 22:39 vmlinuz -> boot/vmlinuz-4.10.0-42-generic
lrwxrwxrwx  1 root root   29 дек  9 15:38 vmlinuz.old -> boot/vmlinuz-4.8.0-36-generic
user@user-virtual-machine:/$ cd
user@user-virtual-machine:~$ ls
examples.desktop
user@user-virtual-machine:~$ ls -l
total 12
-rw-r--r-- 1 user user 8980 дек  9 15:35 examples.desktop
user@user-virtual-machine:~$ _

```

Команда выводит много информации о файлах текущего каталога. Запомните, что если первый символ строки листинга — прочерк «-», это обычный файл. Каталоги отмечены буквой «d» в первой позиции. В этом списке не все файлы. По традиции, файлы, имя которых начинается с точки, по умолчанию команда `ls` не показывает. Обычно это служебные файлы, о них поговорим позже. Чтобы увидеть все файлы каталога, нужна опция `-a`:

```

user@user-virtual-machine:~$ ls -la
total 60
drwxr-xr-x 6 user user 4096 дек 16 16:03 .
drwxr-xr-x 3 root root 4096 дек  9 15:35 ..
-rw----- 1 user user   39 дек 16 16:46 .bash_history
-rw-r--r-- 1 user user  220 дек  9 15:35 .bash_logout
-rw-r--r-- 1 user user 3771 дек  9 15:35 .bashrc
drwx----- 4 user user 4096 дек 16 16:02 .cache
drwx----- 3 user user 4096 дек 16 16:02 .config
-rw-r--r-- 1 user user 8980 дек  9 15:35 examples.desktop
drwx----- 3 user user 4096 дек 16 16:02 .local
drwxrwxr-x 2 user user 4096 дек 16 16:03 .nano
-rw-r--r-- 1 user user  655 дек  9 15:35 .profile
-rw----- 1 root root 1024 дек 14 19:10 .rnd
-rw-rw-r-- 1 user user   66 дек 16 16:03 .selected_editor
-rw-r--r-- 1 user user    0 дек 14 19:08 .sudo_as_admin_successful
user@user-virtual-machine:~$

```


Обратите внимание на файлы с именами «.» и «..». С последним мы уже сталкивались, это ссылка на родительский каталог. А «.» обозначает специальное имя для текущего каталога. Это путевое имя можно использовать в командах.

Например, для запуска программы с именем **myprog**, если она находится в текущем каталоге, можно набрать **./myprog**. Кроме того, имена скрытых файлов в Linux (и других UNIX-подобных операционных системах) начинаются с точки. На картинке выше мы видим такие примеры: **.bashrc** и **.profile**. Команда **ls** их не отображает по умолчанию, но команда **ls -la** отобразит в том числе и скрытые файлы, но не покажет «.» и «..».

Практическая задача по перемещению файлов

В каталоге **~/My best photos** находится большое количество файлов с именами вида **IMG_YYYYMMNN.jpg**, где **YYYY** — числовое четырехзначное значение, представляющее год, **MM** — месяц, **NN** — порядковый номер фотографии. Например, мы их скопировали с флешки. Необходимо создать двухуровневую структуру подкаталогов вида **YYYY/MM** и переместить в них соответствующие файлы. Перед началом работы требуется создать резервную копию **My best photos**.

Резервную копию создаем командой **cp** (CoPy):

```
user@user-virtual-machine:~$ cp -R My\ best\ photos/ My\ best\ photos.bak
```

Опция **-R** предназначена для рекурсивного копирования каталогов. Если внутри содержатся подкаталоги, они также будут скопированы. Обратите внимание на символы «\» перед пробелами в имени каталога. Они необходимы, так как пробел используется для разделения параметров команды. Таким образом имя с пробелами, экранированными обратным слешем, воспринимается командой как один параметр. Как альтернативу для экранирования пробелов можно использовать имя в кавычках. Сделаем каталог **My best photos** текущим:

```
user@user-virtual-machine:~$ cd My\ best\ photos/My\ best\ photos.bak
user@user-virtual-machine:~/My best photos$
```

Каталоги создаются командой **mkdir**:

```
user@user-virtual-machine:~/My best photos$ mkdir 2000/01
mkdir: невозможно создать каталог «2000/01»: Нет такого файла или каталога
user@user-virtual-machine:~/My best photos$
```

Ошибка появилась потому, что для подкаталога **01** отсутствует родительский каталог **2000**. Надо заранее создавать каталог **2000** или использовать опцию **mkdir -p** для автоматического создания родительского каталога.

Можно создать несколько каталогов за один вызов команды:

```
user@user-virtual-machine:~/My best photos$ mkdir 2000 2001 2002
```

Но есть более быстрый вариант создать каталоги с 2000 по 2018 годы:

```
user@user-virtual-machine:~/My best photos$ mkdir 20{00..18}
```

Командный интерпретатор автоматически раскроет фигурные скобки и подставит вместо последнего параметра все значения: 2000 2001 2002 ... 2018.

Фигурные скобки можно использовать с любыми командами, где требуются несколько параметров. Кроме двоеточия, которое позволяет перечислять числовые значения, можно использовать вариант с явным перечислением через запятую, например так:

```
mkdir 200{1,2,3,4,5}
```

Аналогично для быстрого создания каталогов по годам и месяцам с использованием ключа **mkdir -p**:

```
mkdir -p 20{00..18}/{01..12}
```

Теперь для перемещения фотографий в каталоги напишем небольшой цикл с использованием конструкции языка командного интерпретатора (более подробно разберем на следующих занятиях):

```
user@user-virtual-machine:~/My best photos$ for i in {2000..2018}
> do
>   for j in {01..12}
>     do
>       mv IMG_$$j* $i/$j
>     done
> done
```

Каждая строка завершается вводом. Символ > в начале строки добавляется командным интерпретатором в знак того, что конструкция требует продолжения. Мы использовали два вложенных цикла **for** такого типа:

```
for i in {2000...2018} do
... $i ...
done
```

На каждой итерации цикла параметр **i** принимает одно из значений с 2000 по 2018. Чтобы использовать значение параметра в теле цикла, к нему можно обратиться как **\$i**. Для перемещения группы файлов за одну операцию используется команда **mv** в форме:

```
mv file1 file2 ... fileN dir
```

Все параметры команды, кроме последнего — имена перемещаемых файлов. Последний параметр — каталог назначения. Имена файлов формируются с использованием параметров циклов и специального символа «*»: вместо **IMG_\$\$j*** команде **mv** будет передан список всех файлов текущего каталога, которые начинаются с **IMG_YYYYMM**. Каталог назначения формируется на основе значений параметров цикла на данной итерации **\$/j YYYY/MM**.

После того как мы успешно перенесли все файлы в нужные каталоги и убедились, что все правильно, можно удалить резервную копию командой **rm**.

```
user@user-virtual-machine:~/My best photos$ rm -rf ~/My\ best\ photos.bak/
```

Опция **-r** говорит, что надо удалить каталог рекурсивно. Опцию **-f** следует использовать с осторожностью: она сообщает команде, что удаление будет без запросов подтверждения. При удалении файлов в командной строке надо помнить, что в этом случае файлы удаляются окончательно, без возможности восстановления из корзины.

Файлы для подобных экспериментов можно получить с флешки, смонтировав ее с помощью **mount**, или закопировать по **sftp** с помощью любого sftp-клиента — FileZilla, Total Commander или FAR, если вы устанавливали openssh. Если у вас таких файлов нет, можно сделать заглушки с именами.

Например так:

```
touch IMG20{00..18}{00..12}{00..05}.jpg
```

Для проверки переименования можно в каждый файл записать его имя. Но надо понимать, что это не изображение, а просто текстовый файл, названный как картинка и внутри содержащий только оригинальное имя. Сделать это можно так:

```
user@user-virtual-machine:~/My best photos$ for i in *.jpg
> do
> echo $i >$i
> done
```

Вспомогательные клавиши для эффективной работы в командной строке

При работе в командном интерпретаторе **bash** есть несколько комбинаций клавиш, позволяющих существенно облегчить работу. Например, в параметрах команд можно не набирать имена существующих файлов или каталогов целиком, достаточно набрать их начало и нажать клавишу табуляции.

```
user@user-virtual-machine:~$ cd ~/My<TAB>
```

Дальше bash выполнит поиск среди файлов с таким началом и в случае однозначного совпадения имя будет сразу дополнено:

```
user@user-virtual-machine:~$ cd ~/My\ best\ photos/
```

Если найденных вариантов несколько, они все будут показаны:

```
user@user-virtual-machine:~/My best photos$ ls 20<TAB>
2000/ 2002/ 2004/ 2006/ 2008/ 2010/ 2012/ 2014/
2001/ 2003/ 2005/ 2007/ 2009/ 2011/ 2013/
user@user-virtual-machine:~/My best photos$ ls 20
```

Можно продолжить набор, сократив количество вариантов:

```
user@user-virtual-machine:~/My best photos$
ls 201<TAB> 2010/ 2011/ 2012/ 2013/ 2014/
user@user-virtual-machine:~/My best photos$ ls 201
```

И так далее.

Аналогично работает автодополнение по именам команд:

```
user@user-virtual-machine:~/My best photos$ ma<TAB>
mail-lock make-ssl-cert manpath
mawk mail-touchlock man mapfile mail-unlock
mandb mapscrn make manhole mattrib
user@user-virtual-machine:~/My best photos$ ma
```

Кроме автодополнения есть возможность повторять ранее набранные команды с помощью стрелок вверх-вниз на клавиатуре. Это называется историей, или стеком команд.

Можно выполнять поиск по истории команд с помощью комбинации Ctrl-R (поиск назад):

```
user@user-virtual-machine:~/My best
photos$ (reverse-i-search) `':
```

Введите строку поиска — и если в истории команд была команда с такой подстрокой, она будет найдена и подставлена.

```
user@user-virtual-machine:~$ cd ~/My\ best\ photos/

(reverse-i-search)`cd': cd ~/My\ best\ photos/
```

Можно нажать Enter для немедленного выполнения команды или <ESC>, чтобы найденную команду можно было отредактировать.

Кроме того, существуют команды для быстрого перемещения по командной строке и быстрого удаления текста:

- Ctrl-A — в начало строки.
- Ctrl-E — в конец строки.
- Alt-F — на слово вперед.
- Alt-B — на слово назад.
- Ctrl-U — удалить в строке все символы от текущей позиции до начала строки.
- Ctrl-K — удалить в строке все символы от текущей позиции до конца строки.

И еще четыре важных для работы в терминале комбинации:

- Ctrl-S остановит вывод на терминал. Иногда комбинацию используют при выводе больших объемов текста на терминал для временной остановки вывода. Бывает, что эту комбинацию нажимают случайно, и тогда терминал как будто зависает. Не спешите паниковать, просто выполните следующую комбинацию.
- Ctrl-Q продолжает вывод после Ctrl-S. Если кажется, что терминал завис, в первую очередь попробуйте эту комбинацию.
- Ctrl-C завершает работу текущей программы. Используют, когда надо быстро завершить работу текущей программы или скрипта. Программа не всегда может завершиться после Ctrl-C: иногда реакцию на эту комбинацию в программе или скрипте намеренно блокируют.
- Ctrl-R — для поиска по истории команд.

Если на первых порах работать с командной строкой в терминале тяжело, можно установить консольный файловый менеджер — **mc** (Midnight Commander). Это файловый менеджер для текстовых терминалов с текстовыми окнами, меню и поддержкой функциональных клавиш. Кроме того, **mc** содержит простой и удобный редактор для текстовых файлов — **mcedit**. Работа в **mc** не требует дополнительного обучения. Единственная тонкость может быть связана с поддержкой функциональных клавиш в разных моделях терминалов. Если функциональные клавиши не работают, используйте комбинации вида <ESC>1 вместо F1, <ESC>2 вместо F2 и так далее.

Редакторы и создание файлов

Простейшие способы создания файлов

Во многих языках программирования есть команда, с помощью которой можно напечатать на экран строку. Такая команда в оболочке Linux — **echo**. При этом не обязательно обрамлять печатаемую строку в кавычки.

```
echo Hello world
```

С помощью этой же команды можно создать простейший текстовый файл.

```
echo Hello world > hello.txt
```

Перед этим убедитесь, что вы находитесь в домашней директории.

Если вы установили **mc**, можно редактировать и создавать файлы с помощью **mcedit**.

Так, чтобы создать новый файл **newfile.txt**, достаточно набрать:

```
mcedit newfile.txt
```

Подсказки в нижней части экрана указывают, какие действия можно сделать. Esc — для выхода. Если вы вносили изменения, редактор спросит, сохранять или нет.

Можно создать файл с консольного ввода:

```
cat > file2.txt
```

Вы можете вводить строки, нажимая Enter, — будет осуществляться ввод следующей строки.

Чтобы выйти, необходимо послать сигнал конца файла — Ctrl-D (^D). Такой способ может быть удобен при работе с PuTTY, так как вы можете скопировать конфигурационный файл из буфера обмена Windows в создаваемый файл (в примере — **file2.txt**).

Также есть редактор vi, сложный для начинающих. Одна из самых больших проблем — как из него выйти. Для этого нажмите : и наберите q! и Enter. А теперь поподробнее.

Создание и редактирование файлов

Пустой файл можно создать командой:

```
$ touch newfile
```

Основная задача **touch** — обновить время последнего изменения файла, которое показывает **ls -l**. Создание пустого файла, если он не существует, — приятный бонус команды **touch**.

Но файл можно создать и с помощью редактора. Вызывая редактор с именем файла, вы правите существующий файл или создаете еще не существующий.

Наиболее часто применяются редакторы:

- **mcedit**;
- **nano**;
- **vi (vim)**.

mcedit

Это, вероятно, самый простой редактор для тех, кто привык к Norton Commander.

Правка осуществляется прямо в окне редактирования, доступны команды:

- **Esc** — выйти. Спросит, надо ли сохранить, или нет;
- **F2** — сохранить;
- **F3** — начать выделение / закончить выделение;
- **F5** — скопировать выделенный текст;
- **F6** — переместить выделенный текст;
- **F7** — поиск подстроки;
- **F8** — удалить выделенный текст;
- **F9** — меню (доступны копирование в специальный буфер и вставка из него).

Входит в состав **mc**, то есть требуются права на установку дополнительных программ.

nano

Это очевидно — самый простой редактор, доступный в Ubuntu изначально (в **minimal**-установках его также необходимо будет доставить самостоятельно).

В **nano** нет разделения на редактирование и просмотр, что более привычно для пользователей Windows. Кроме того, внизу экрана есть две справочные строки с самыми важными командами редактора. У **nano** все команды начинаются со значка **^**, что означает клавишу **Control**.

Например:

- **^X** (т.е. **Ctrl-X**) или **F2** — выйти из **nano** (спросит, сохранить ли файл);
- **^O** (т.е. **Ctrl-O**) или **F3** — записать файл на диск (спросит, под каким именем сохранить. По умолчанию будет указано то, которое было при запуске, но его можно изменить);
- **^W** (т.е. **Ctrl-W**) или **F6** — поиск с регулярным выражением (с ними мы еще познакомимся).

vi

Непривычный для многих редактор. Обладает двумя режимами работы, команд и редактирования.

Это классический и мощный редактор, который есть в любой UNIX-подобной системе. Но он не является интуитивно понятным и требует минимальной подготовки. Многие начинающие пользователи сталкивались с проблемой, как из него выйти (:q!).

Есть ряд причин, по которым необходимо хотя бы минимальное знакомство с vi:

- vi есть в любом UNIX, папо не слишком распространен в коммерческих системах;
- возможно, это будет единственный редактор в ситуации, когда вы загрузились с аварийного CD и пытаетесь починить систему;
- vi незаменим в экстремальных ситуациях, когда система перегружена и реакция на нажатие клавиши может составлять десятки секунд;
- клавиатурные комбинации из vi можно эффективно использовать для редактирования команд в оболочке и перемещения по истории команд;
- это правда очень мощная штука, некоторые программисты даже используют ее как IDE.

Главная особенность vi, которая вызывает недоумение у новичков: редактор действует в двух режимах — просмотра (режим команд) и редактирования текста. В режиме просмотра нельзя редактировать, в режиме редактирования нельзя перемещать курсор по файлу. Это неудобство призвана исправить улучшенная версия vi — vim, или VI iMproved (улучшенный vi). В нем в режиме редактирования работают стрелки для перемещения по тексту.

Обычно, как только вы загружаете редактор vi или vim, он сразу оказывается в режиме просмотра: вы можете водить курсором, искать текст, выполнять перемещения по файлу. Но не изменять текст файла. Для этого надо переместить курсор в нужную позицию и использовать одну из команд, переводящих vi в режим редактирования (например, i для ввода с позиции курсора).

В оригинальном vi, когда вам потребуется снова переместить курсор, надо опять сменить режим vi с редактирования на просмотр (клавишей ESC) и использовать клавиши перемещения: стрелки или (так как в терминале они могут не работать) их буквенные аналоги — j, k, h, l (вниз, вверх, влево, вправо). Потом опять редактирование, вновь перемещение и т. д.

Когда вы закончили работу, надо сохранить файл. Для этого в режиме просмотра надо набрать - :w <filename> <ENTER> filename и имя файла, где будут сохранены изменения. Если у файла уже есть имя, можно не указывать filename. Для выхода из vi набрать в режиме просмотра - :q. Если надо выйти без сохранения изменений — - :q!

Как найти слово в vi

Чтобы быстро найти слово в vi (vim), просто нажмите знак / или ?, а затем напишите слово, которое ищете. Когда нужное слово найдено, можете использовать клавишу n, чтобы перейти непосредственно к его следующему положению в тексте.

vi (vim) также позволяет запускать поиск по слову, на которое указывает курсор мыши. Для этого разместите курсор на нужном термине (слове), а затем нажмите * или #, чтобы просмотреть все его расположения.

Основные элементы Linux

Разберем основные компоненты Linux-системы более подробно: ядро, графическую оболочку, командный интерпретатор, UNIX way. Но для начала остановимся на философии UNIX, которой следуют UNIX-подобные операционные системы, в том числе Linux.

UNIX way

Философия Unix, известная как UNIX way, — это подход, которому следует архитектура UNIX-подобных систем. Есть несколько разных определений, что такое UNIX way. Среди них можно выделить несколько важных принципов:

- утилиты — маленькие отлаженные программы, решающие только свою задачу;
- более сложные задачи можно решать, комбинируя утилиты, запуская их с помощью конвейеров;
- конфигурационные файлы хранятся в виде простых текстовых файлов.
- всё есть файл.

Это означает, что лучше одна программа, решающая только одну задачу, но делающая это хорошо. Благодаря использованию потоков и возможности работы с устройствами как файлами, вывод и ввод таких программ можно совмещать, добиваясь гибкости. Мы уже использовали это на примере комбинации команды **apropos** и **grep**.

Принципы UNIX вы будете использовать при работе Linux и наблюдать постоянно.

Ядро

Центральный элемент любой операционной системы — ядро. Ядро загружается в память при старте системы, и его код выполняется процессором на привилегированном уровне (по сравнению с пользовательскими процессами). Для пользователя эта часть системы прозрачна и не очень заметна. Зато программисты активно используют функции ядра, или системные вызовы, при написании программ, которые работают с файлами и периферийным оборудованием. Ядро взаимодействует с оборудованием посредством встроенных или динамически подгружаемых драйверов и позволяет программистам работать с периферией как с обычными файлами.

Доступ к файлам и файловым системам — также функция ядра. Вообще унифицированное понятие файла (файлом является почти любое устройство: жесткий диск, мышь, звуковая карта) — характерная черта систем семейства UNIX, составляющая вместе с остальными ключевыми особенностями стиль UNIX way.

Поскольку Linux является многозадачной и многопользовательской системой, ядро также занимается распределением ресурсов между программами и обеспечивает разграничение доступа к ресурсам. Например, попытка процесса (программы на этапе исполнения) обратиться за пределы своего адресного пространства немедленно приведет к его аварийному завершению системой без ущерба для остальных процессов. Переключение между процессами и диспетчеризация ввода-вывода — также важные функции ядра.

Системные утилиты, оболочка

Кроме ядра важной частью системы являются системные утилиты. В Linux они представлены стандартным набором утилит, разработанных в рамках проекта GNU. Среди них есть команды администрирования пользователей, утилиты для управления файлами и процессами, а также утилиты для обработки текста.

Большинство утилит не имеет графического интерфейса. Они выдают и принимают на вход информацию в виде обычного текста, поэтому в UNIX существует множество стандартных утилит для обработки и форматирования текстов. Для запуска системных утилит и других программ служит командный интерпретатор (его еще называют оболочкой — shell). Это основной инструмент системного администратора Linux и своего рода язык программирования, позволяющий создавать сложные сценарии (скрипты). Также он используется интерактивно для последовательного ввода команд.

Один из принципов стиля UNIX way: каждая утилита предназначена для выполнения отдельной узкой задачи и способна выполнять ее максимально гибко при помощи большого набора опций. А еще утилиты способны взаимодействовать, последовательно передавая друг другу результаты своей работы через конвейер. Это одна из наиболее мощных составляющих UNIX way, которую мы подробно разберем далее.

Демоны

Кроме системных утилит есть класс специальных программ — демонов. Типичный демон — программа, которая обслуживает сетевой сервис. Обычно она уже запущена и находится в ожидании запросов. Примеры демонов, с которыми мы познакомимся дальше: веб-сервер, почтовый сервер, сервер доменных имен. Обратите внимание, что здесь слово «сервер» используется в значении «программа, предоставляющая сервис».

Приложения пользователя, графический интерфейс X Window System

Пользовательские приложения — самый многочисленный класс программ. Это прикладные программы: офисные приложения (LibreOffice), браузер (Mozilla Firefox), почтовый клиент (Thunderbird).

Прикладные программы не являются частью системы, поэтому их набор и версии могут отличаться для разных дистрибутивов.

Графические программы используют систему графического интерфейса X Window System. Графическая система появилась в UNIX с середины 80-х. В отличие от графической среды Microsoft Windows, в UNIX графическая среда не является частью ядра системы — это отдельный набор программ и библиотек, от которого можно отказаться. Обычно на серверах вообще не используют графическую оболочку, выполняя все задачи с использованием интерпретатора команд. X Window System изначально задумывалась как сетевая система клиент-серверного типа. Это подразумевает физическое разделение клиентов:

- программ, которые отправляют запросы на вывод графики,
- и X-сервера — программы, которая запущена на стороне пользователя и непосредственно занимается выводом графики на графическом устройстве клиента, обслуживая запросы X-клиентов на отрисовку графических примитивов.

Это необычное расположение клиента и сервера: сервер работает на компьютере пользователя, а клиенты находятся на удаленной системе. Клиент-серверная архитектура X Window и наличие открытого стандартного протокола позволяет запускать графические приложения Linux на широком круге устройств и операционных систем. X-сервер как стандартное приложение существует почти в любой Unix/Linux системе. X-серверы также доступны для ОС семейства Windows. Linux предоставляет множество графических окружений, хотя почти все они основаны на X Windows. Их разнообразие проявляется в компонентах графической подсистемы: оконных менеджерах и графических средах. Кратко остановимся на самых распространенных.

X-Window и оконные менеджеры

Оконные менеджеры — программы, которые управляют размещением на рабочем столе окон вновь запущенных приложений и позволяют менять их размер, перетаскивать, схлопывать и разворачивать. Оконные менеджеры определяют внешний вид окон, оконное меню, рамки, кнопки управления, тени и так далее. Наиболее популярные менеджеры окон — Enlightenment, FVWM, Openbox.

Как правило, оконные менеджеры используются в составе окружений рабочего стола. Это набор специальных приложений для организации рабочего стола: панели запуска, меню доступа к приложениям, индикаторы событий, специальные поисковые элементы.

Еще в состав окружения обычно входит набор приложений пользователя, плотно интегрированных с остальными элементами. Современные окружения рабочего стола максимально задействуют ресурсы графического адаптера и сопровождают работу пользователя 3D-анимацией и различными визуальными эффектами.

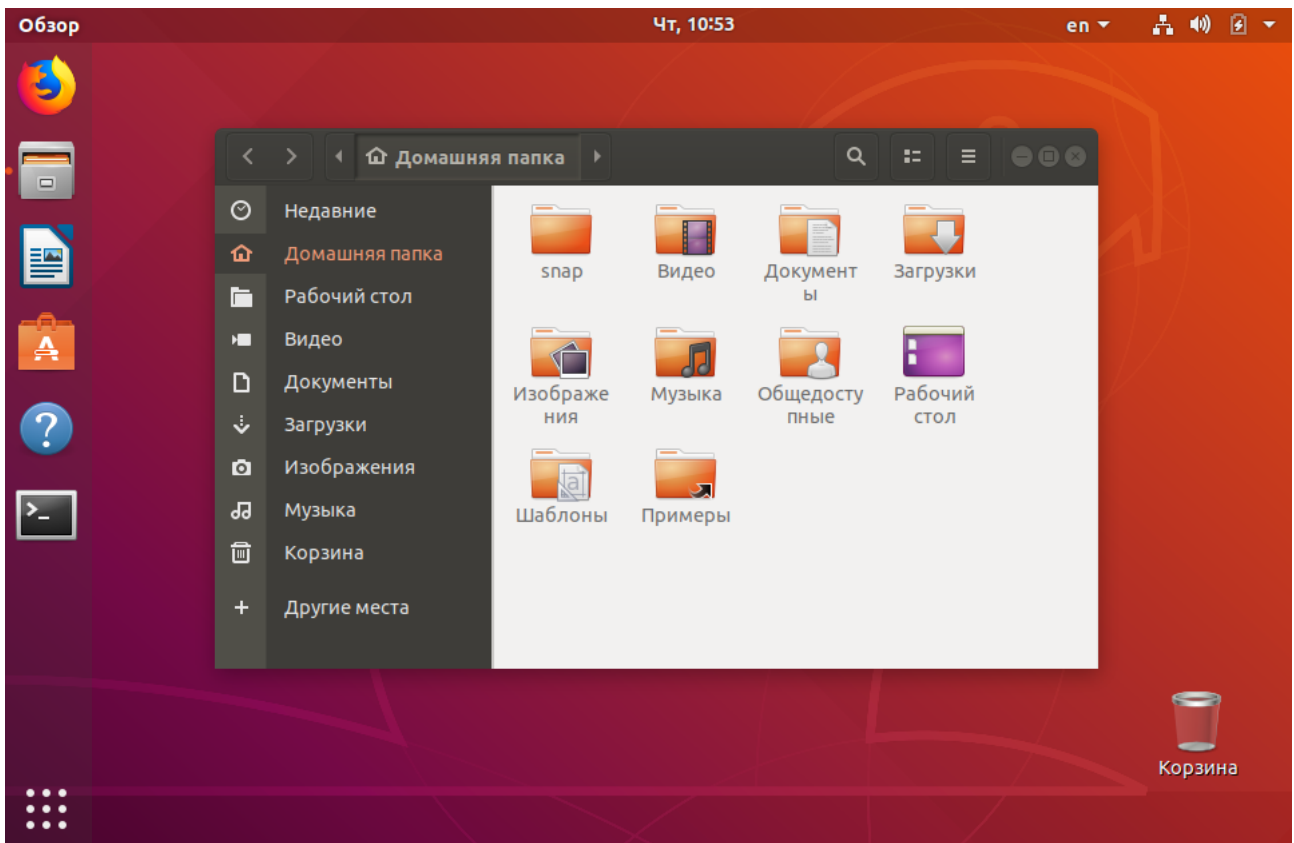
Среди популярных окружений стоит отметить Unity, KDE4, Gnome 3. Первая использовалась как основная до 17-й версии, а последняя теперь используется в основной десктоп-версии дистрибутива Ubuntu 18.

Помимо основной версии, сообщество разработчиков Ubuntu поддерживает специальные дистрибутивы, основанные на альтернативных графических средах — KDE, XFCE и LXDE. Среди популярных альтернативных сборок Ubuntu: Kubuntu (с KDE), Xubuntu (XFCE) и Lubuntu (LXDE). Если на вашем компьютере мало ресурсов, лучше вместо основной версии Ubuntu использовать дистрибутив Lubuntu, использующий графический интерфейс LXDE, которому достаточно для работы 512 Мб RAM. LXDE использует привычную концепцию меню программ в стиле Windows XP, поэтому мы не будем на нем останавливаться.

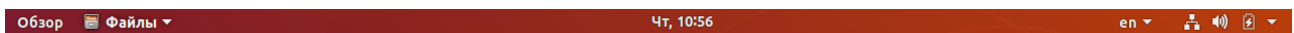
В курсе мы кратко расскажем о работе с графическим менеджером Ubuntu. Эта графическая среда может показаться непривычной после интерфейса Microsoft Windows или других графических сред Linux, поэтому далее рассмотрим основные элементы рабочего стола и пример пользовательской сессии.

Основные элементы рабочего стола

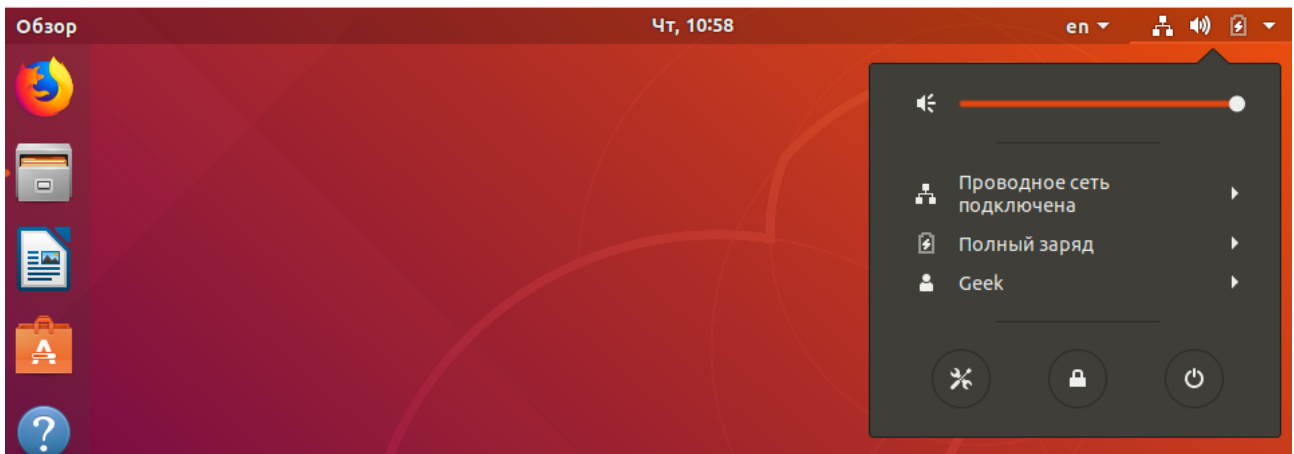
После ввода пароля и регистрации в системе пользователь получает картинку рабочего стола. Кратко расскажем о его основных его элементах.



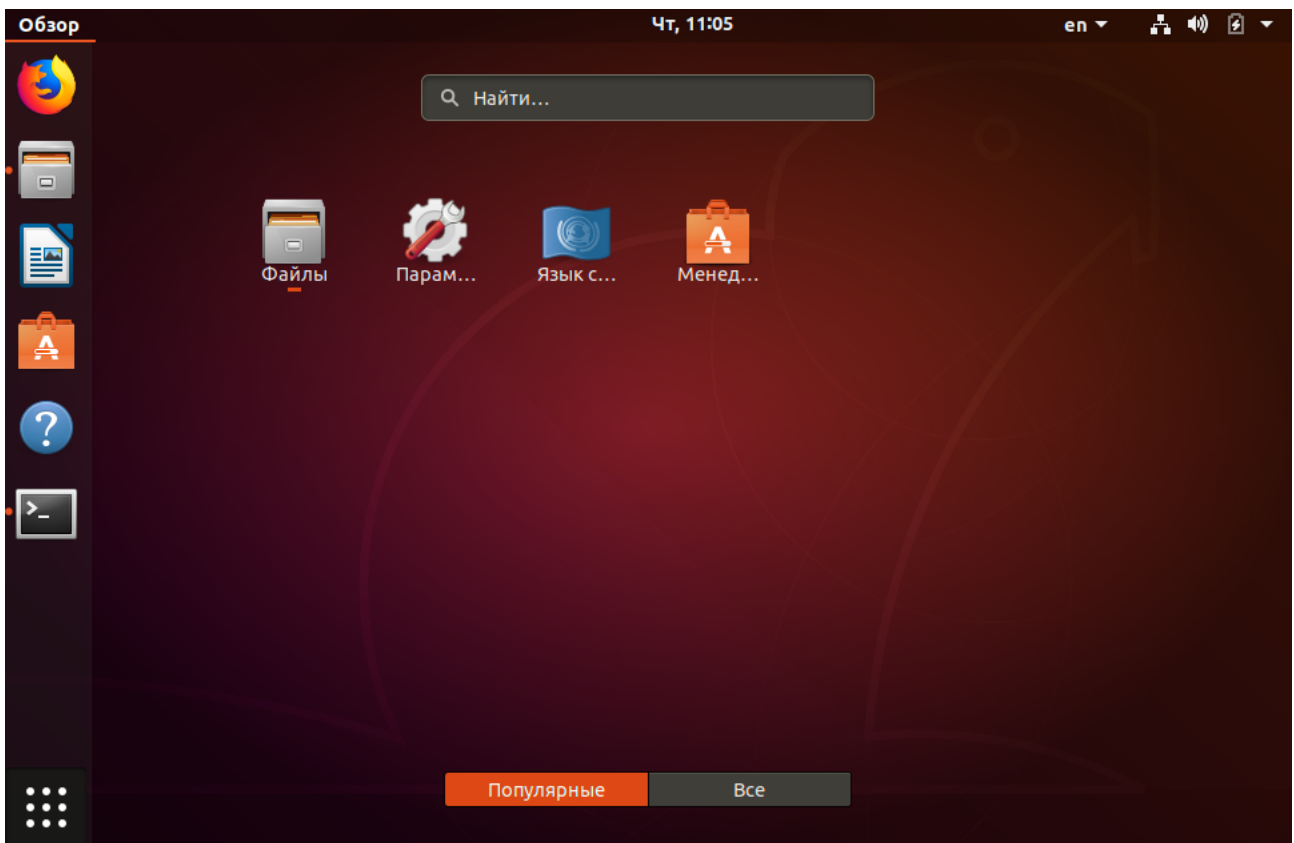
Панель запуска — набор значков вдоль левой стороны экрана для запуска часто используемых приложений. Запуск приложения выполняется по однократному клику на значок. При запуске значок начинает пульсировать, а стартовавшее приложение отмечается маленьким красным кружком слева от значка запущенного приложения. Если приложение запущено в нескольких экземплярах, кружков слева будет столько же, сколько экземпляров (максимально отображается до четырех). По щелчку правой кнопкой мыши на значке появляется контекстное меню приложения и список уже запущенных экземпляров.



Верхняя панель — это горизонтальная полоса вдоль верхней части экрана. В левой части отображается меню текущего запущенного приложения; здесь же находятся кнопки управления размером окна, если приложение запущено в полноэкранном режиме. Меню приложения появляется, только если указатель мыши в верхней части экрана. Эта особенность часто ставит в тупик неподготовленного пользователя, который не может найти меню приложения. В правой части меню располагаются индикаторы: индикатор сети, переключатель раскладки, уровень громкости, уровень заряда батареи и так далее. Самый правый индикатор — доступ к настройкам рабочего стола и меню выключения системы.

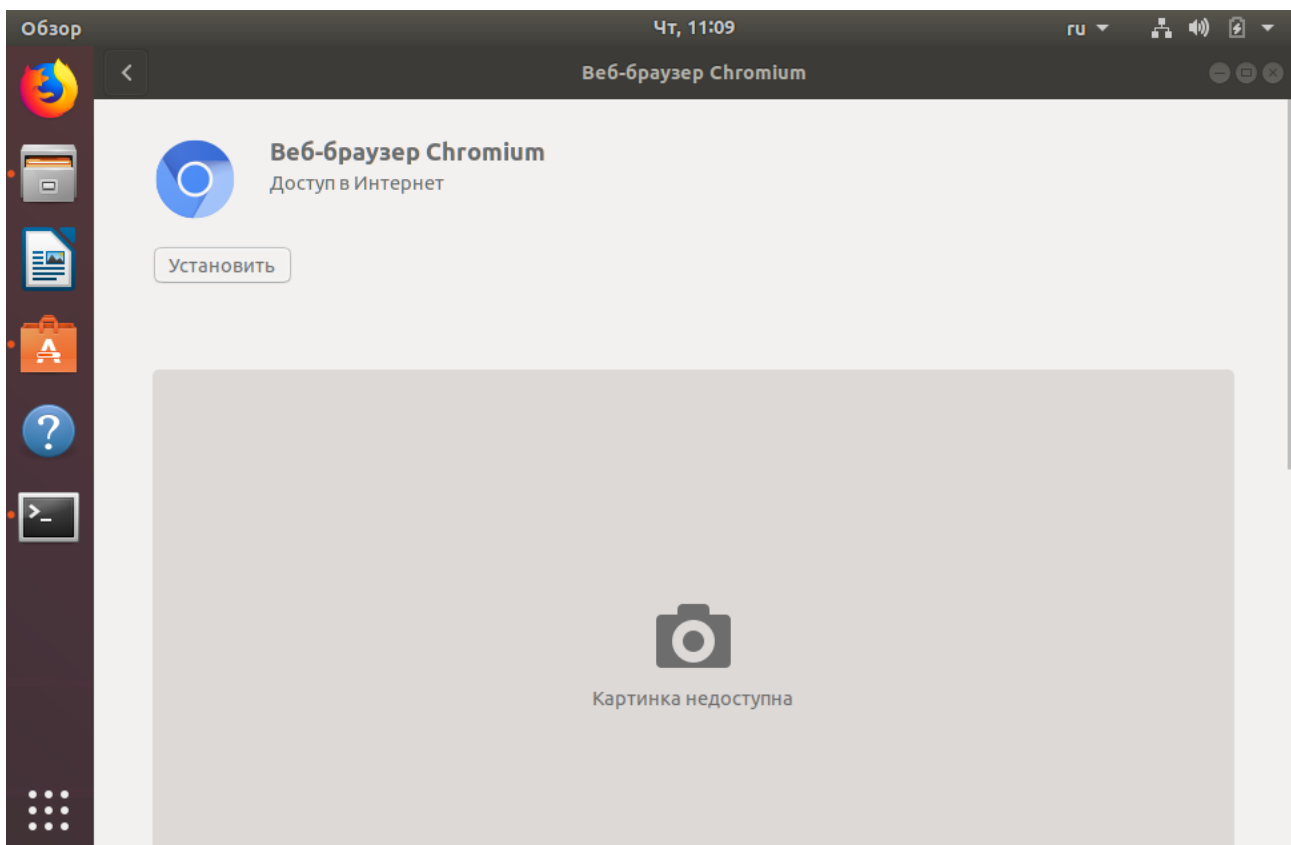


Главное меню — окно, которое появляется по клику вверх на иконку с 9 точками внизу панели запуска. Если после этого ничего не набирать и не щелкать, на экране появятся значки последних запусавшихся приложений и недавно использовавшихся файлов. Закрывается окно главного поиска клавишей ESC.

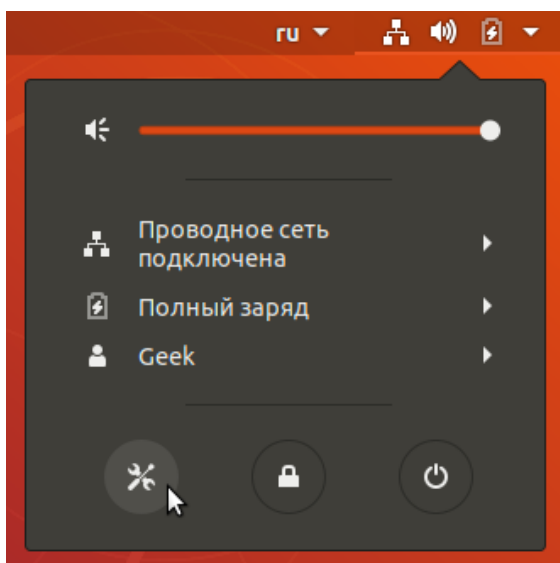




С помощью главного меню можно выполнять поиск установленных приложений и файлов. В нижней части экрана главного меню расположены значки, представляющие не только установленные приложения, но и те, которые можно установить через менеджер приложений Ubuntu.

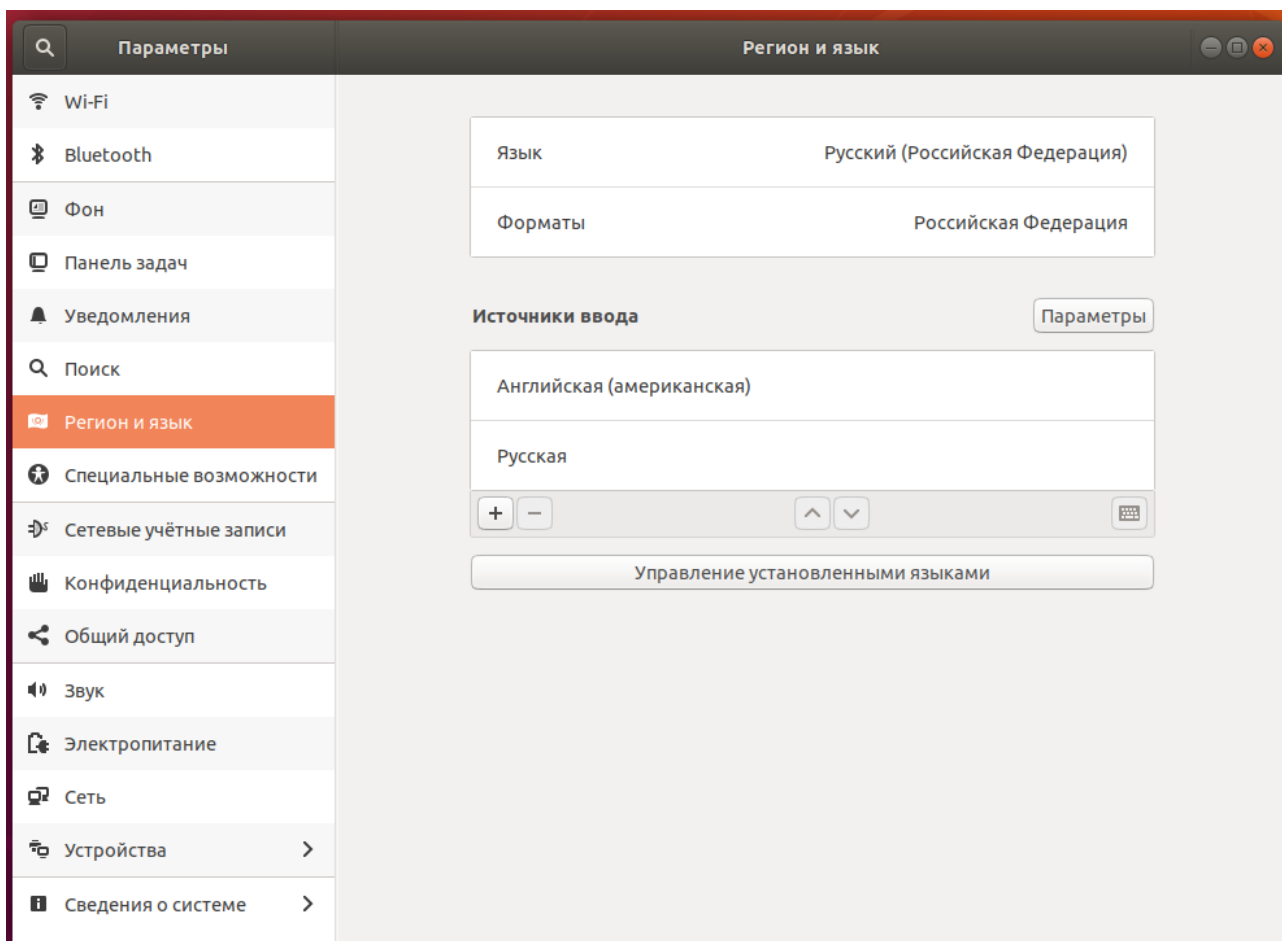


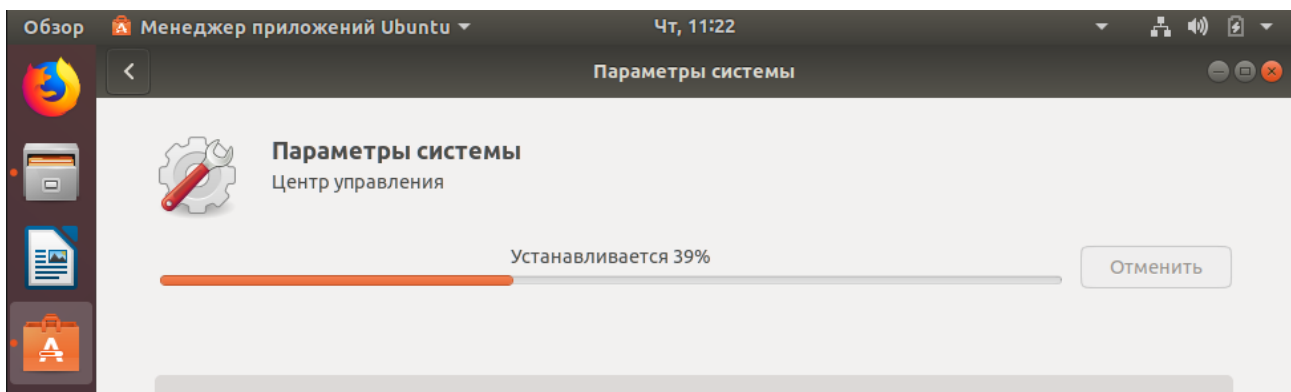
Кликнув по значку каждого приложения, можно получить его краткое описание — и тут же установить приложение.



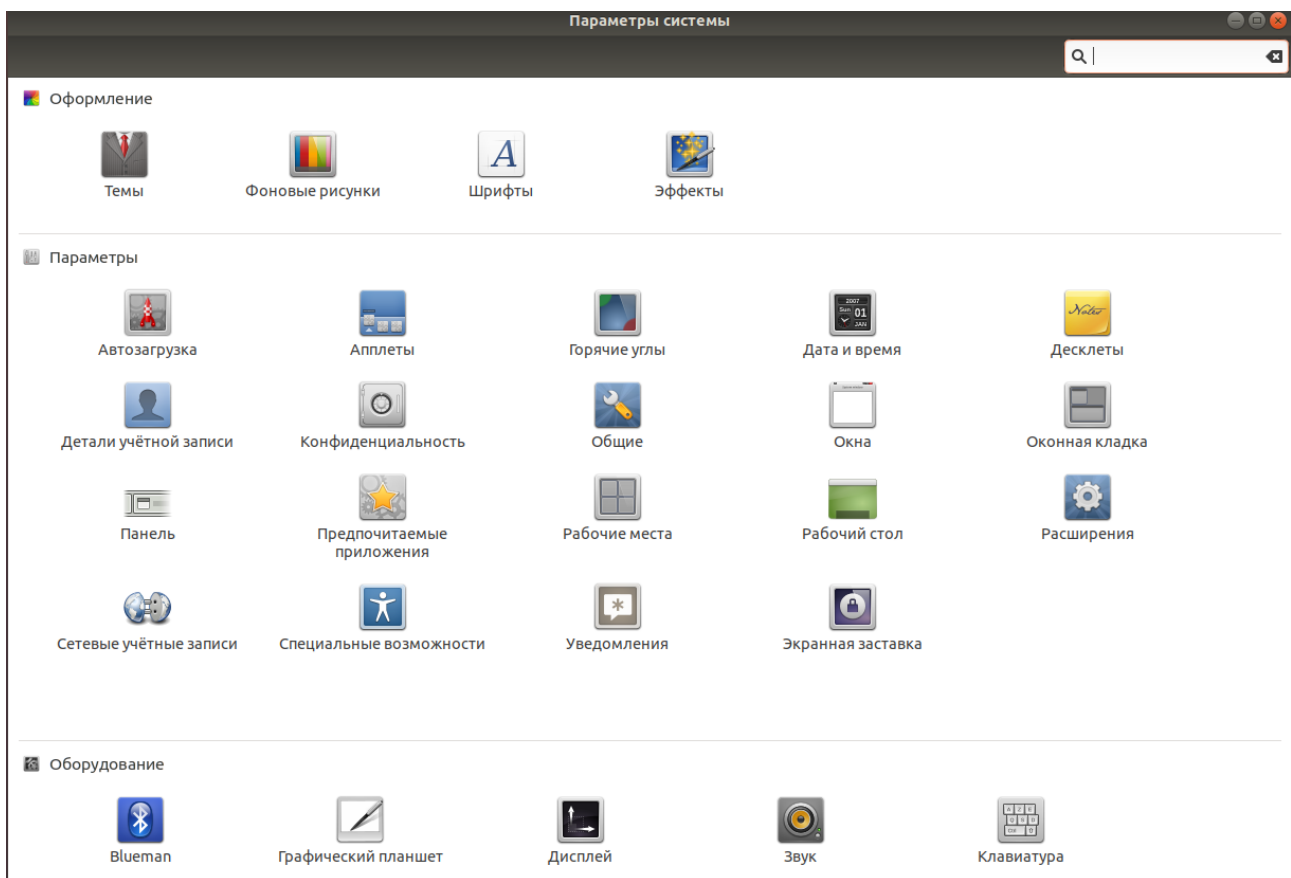
Доступ к базовым настройкам системы можно получить через крайний правый значок глобального меню, выбрав нижнюю левую иконку в виде набора инструментов.

Доступ к графическим утилитам администрирования можно получить, установив параметры системы из глобального меню (набрав там «Параметры системы»):





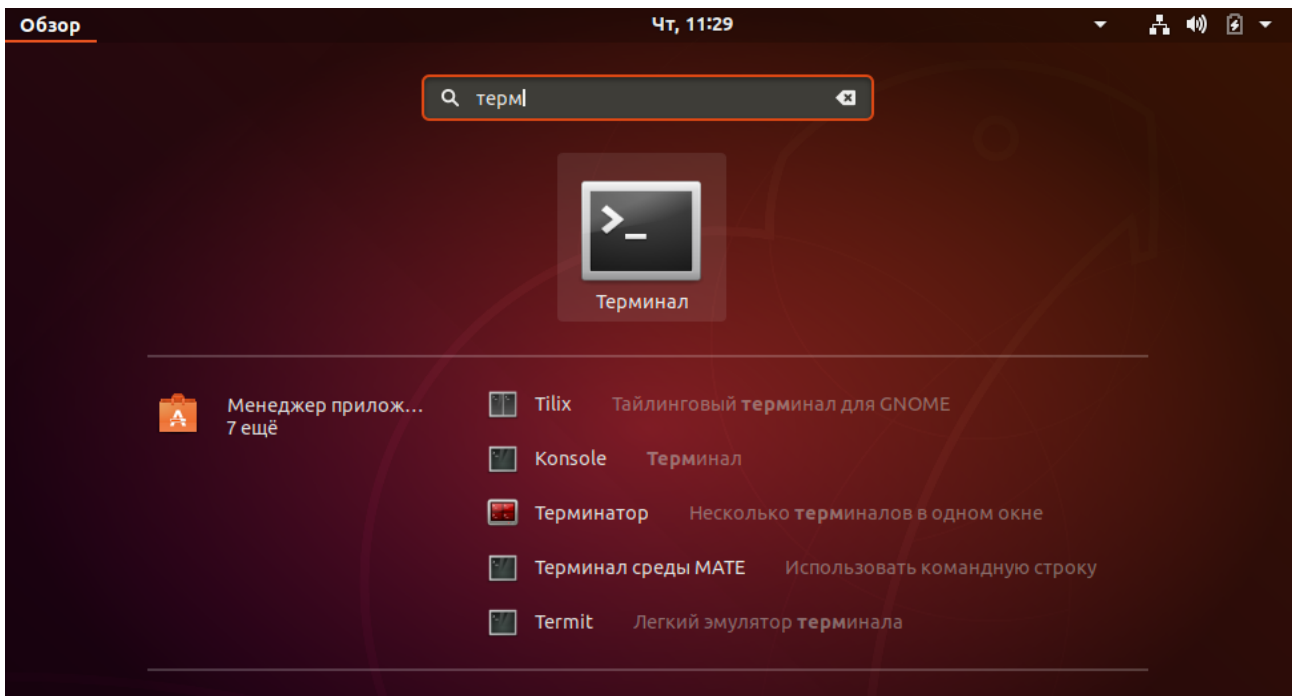
После запуска откроются параметры системы:



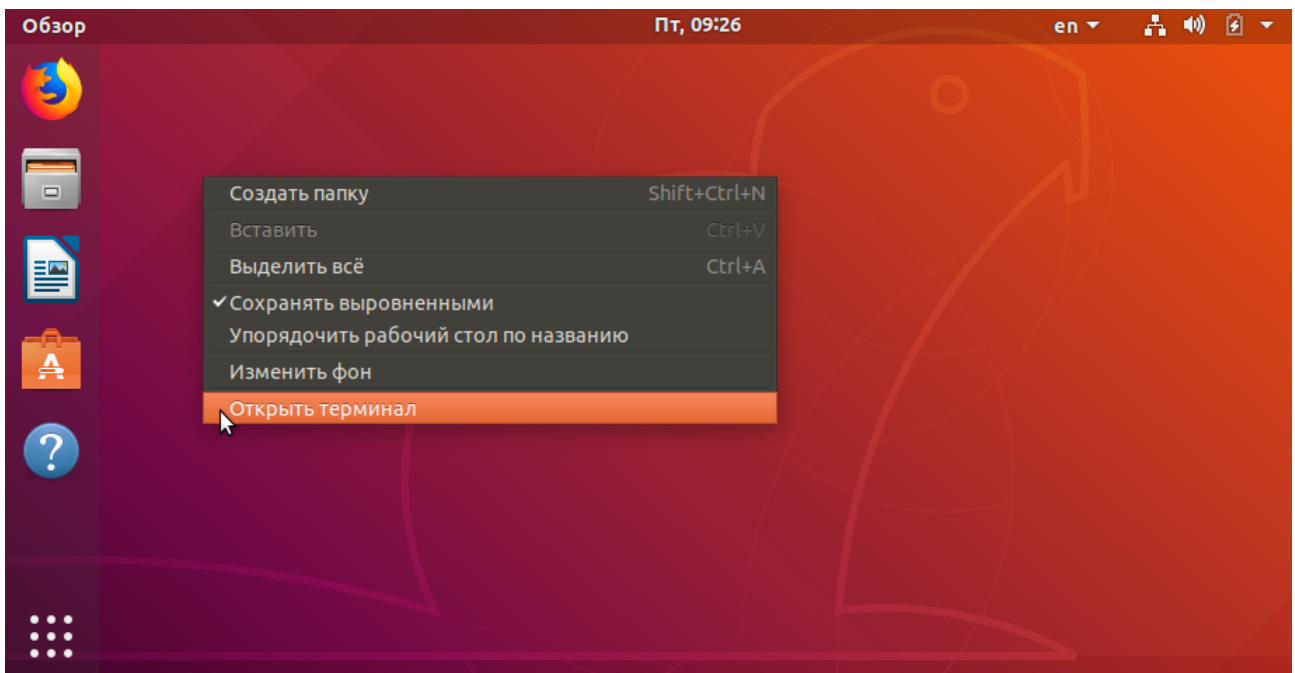
Пример запуска терминала

Рассмотрим на примере запуск приложения «Терминал», которое по умолчанию установлено, но не имеет значка в панели запуска. Мы покажем, как добавить его туда. Далее изменим настройки оформления рабочего стола, добавив к панели запуска значок с рабочими пространствами, и покажем, как ими пользоваться. По завершении сессии выключим систему.

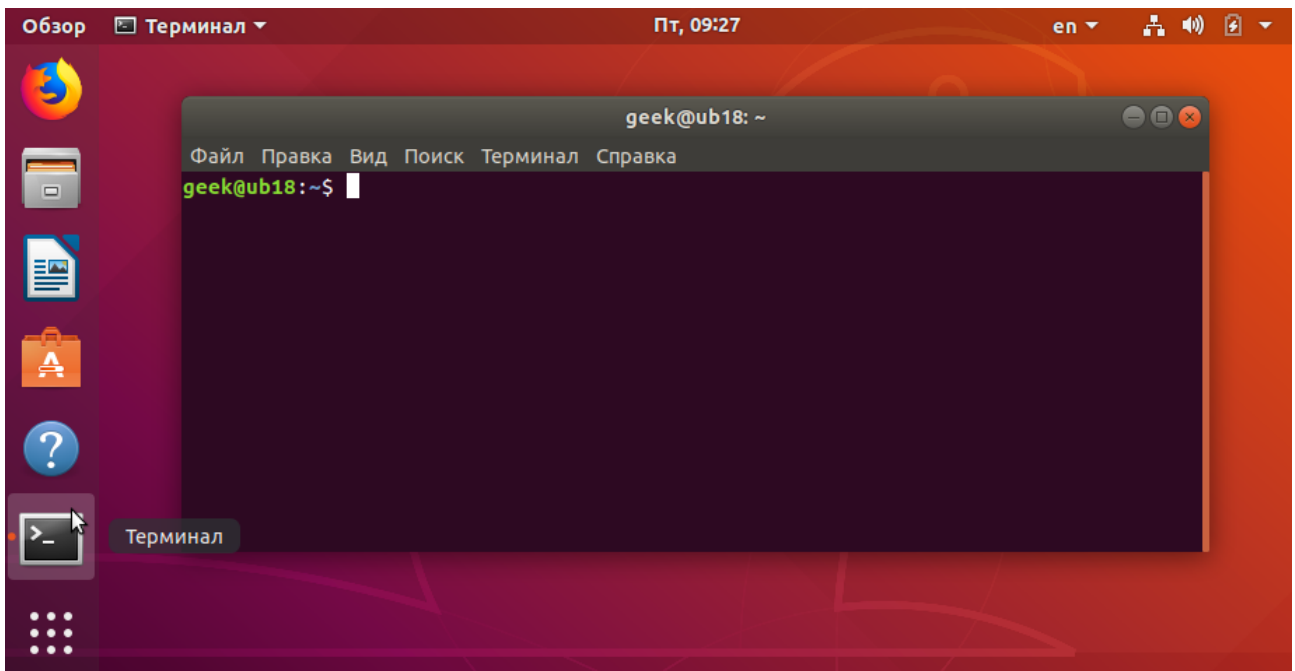
Выполним поиск приложения через главное меню по строке «терм» или «term», появятся варианты установленных приложений и предложения по установке новых. Запустим найденное приложение — «Терминал»:



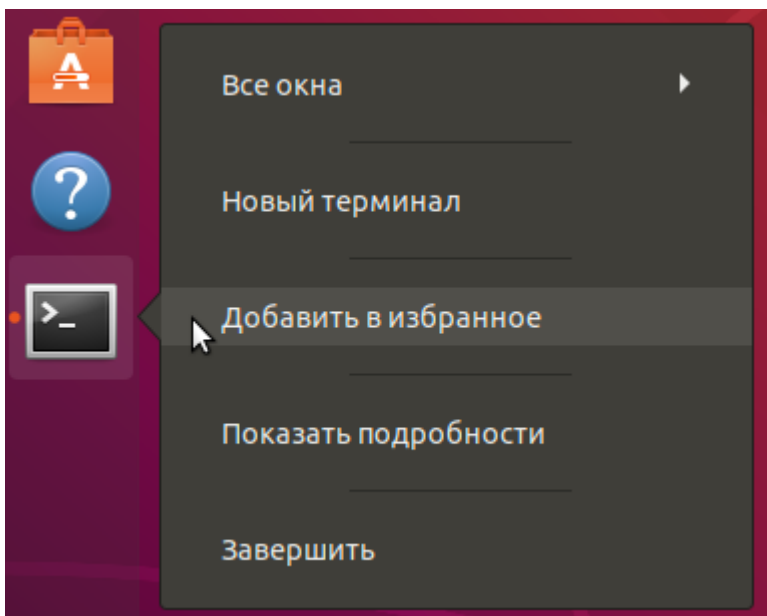
Другой способ запуска терминала — нажать правой кнопкой мыши на любом свободном участке рабочего стола и выбрать из выпадающего списка пункт меню «Открыть терминал»:



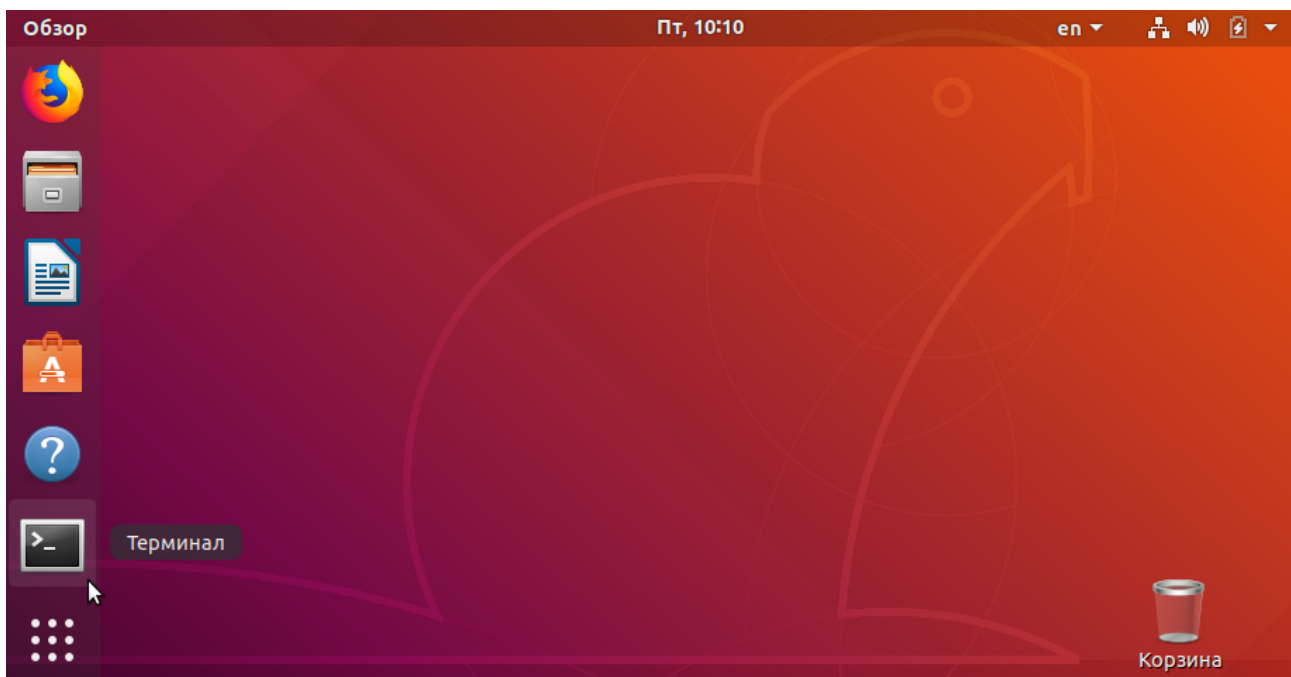
Появится окно терминала — и значок запущенного приложения возникнет в панели запуска слева:



Щелкаем правой кнопкой мыши на значке терминала в панели запуска — появится меню. В нем выберем пункт «Добавить в избранное»:



Теперь терминал можно запускать непосредственно через панель запуска:



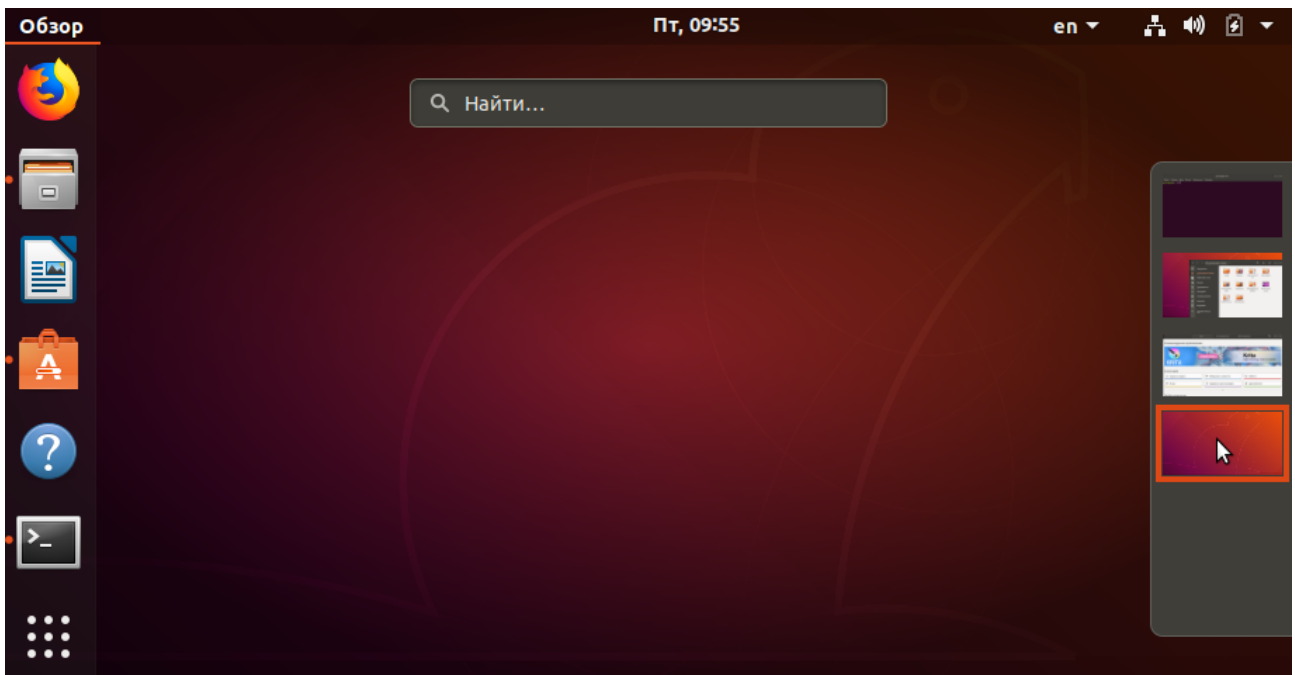
Рабочие столы

Рабочие места используют, чтобы упорядочить окна на рабочем столе. Можно создать несколько рабочих мест, действующих как виртуальные рабочие столы.

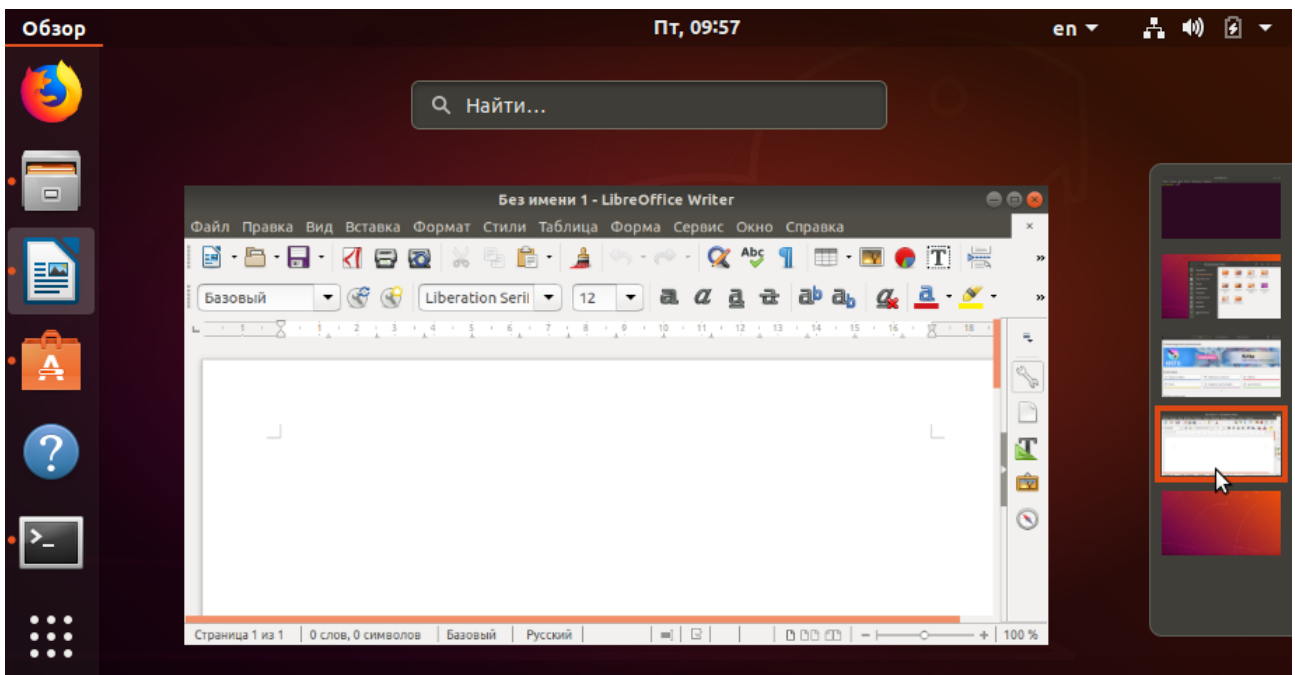
Рабочие места можно использовать для организации работы. Например, собрать все окна коммуникационных программ (почтовой программы или клиента сети мгновенных сообщений) на одном рабочем месте, а окна, необходимые для работы, — на другом. На третьем рабочем месте может быть открыт ваш музыкальный проигрыватель.

Использование рабочих мест

Переведите мышку в верхний левый угол и нажмите «Обзор» — или нажмите клавишу Win на клавиатуре. Справа появится вертикальная панель, показывающая используемые рабочие столы и еще одно дополнительное свободное рабочее место. Это выбор рабочих столов:



Чтобы добавить рабочий стол, перетащите любое запущенное приложение в пустое окно на панели выбора рабочих мест:



Теперь этот рабочий стол содержит запущенное приложение, и появилось новое свободное рабочее место.

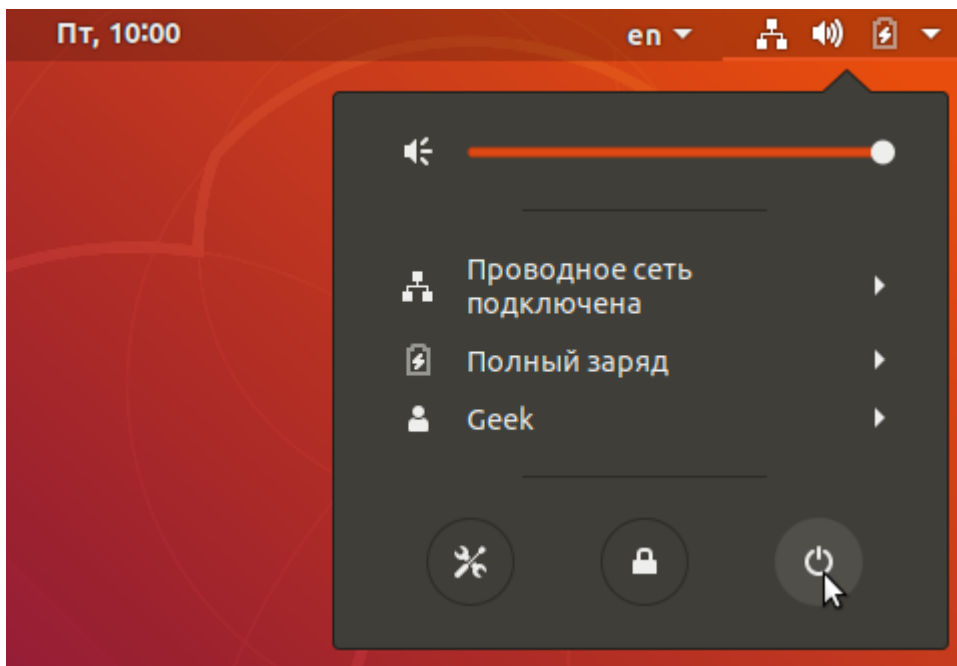
Чтобы удалить рабочее место, просто закройте все открытые на нем окна или переместите их на другие рабочие места.

Всегда доступно по крайней мере два рабочих места.

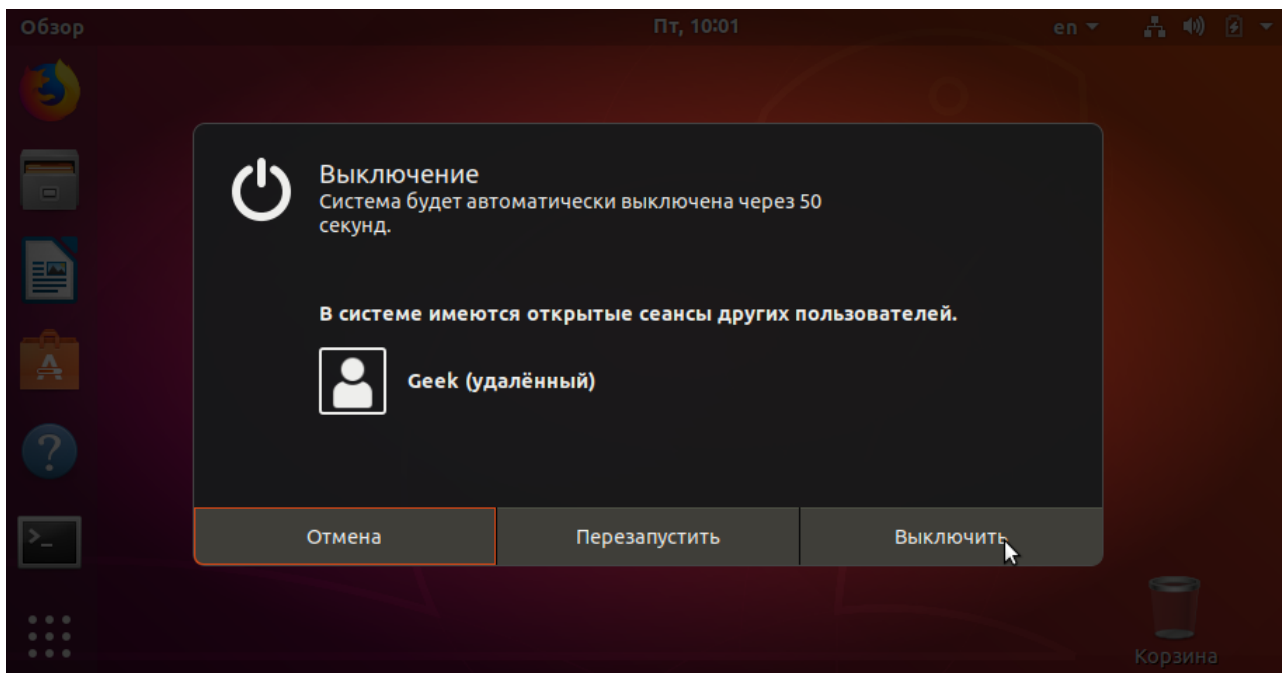
Подробнее вы можете почитать об этом на странице [официальной документации Ubuntu](#).

Выключение системы

Выключим систему через правый индикатор глобального меню кнопка «Выключение»:



Откроется пункт меню «Выключение» — нажмем кнопку «Выключить»:



Процессы, потоки, потоки ядра

Современные операционные системы состоят из как минимум двух частей, сильно отличающихся по своим возможностям:

- ядро, работающее в пространстве ядра и на уровне режима работы процессора взаимодействующее с оборудованием и ресурсами;

- и пользовательское пространство, работающее в уровне ядра, и на уровне режима работы процессора ограниченное в доступе к ресурсам. Попытки обратиться к неразрешенным ресурсам приводят к вызову процессором исключения и передаче управления ядру операционной системы.

В системе Linux многозадачность реализована как для ядра операционной системы, так и для пользовательского пространства. В ядре работают потоки (процессы) ядра. А в пользовательском пространстве используются процессы и потоки.

Многие слышали, что в Linux есть потоки, которые позволяют в одной программе реализовывать параллельное выполнение нескольких функций. Технически потоки реализованы в Linux на базе процессов, но процесс все же является более независимой единицей.

Программисты редко прибегают к методике распараллеливания одной программы при помощи процессов. Но суть в том, что в Unix-системах разработчик обладает полной свободой выбора стратегии многозадачности. И это здорово!

В Linux потоки также являются особым рода процессами (так как они тоже имеют собственные PID), но при этом они делят одну и ту же память процесса, которому они принадлежат: первый созданный процесс — фактически первый поток процесса. При создании копии процесса (fork) происходит копирование его ресурсов. Так что это более безопасно, но менее эффективно.

Каждый процесс, кроме упомянутых UID и GID, также имеет идентификаторы PID (process ID) и PPID (parent process ID).

Утилита ps (Process Show)

Перечень процессов можно посмотреть с помощью команды:

```
$ ps aux
```

ps отображает процессы пользовательского пространства и потоки ядра (отображаются в квадратных скобках). Чтобы посмотреть потоки, используйте ключ **-T**, а для потоков конкретного процесса — ключ **-p**.

```
$ ps -T -p 4242
```

Чтобы посмотреть дерево запущенных процессов, можно воспользоваться командой:

```
$ pstree
```

Существуют и другие способы мониторинга процессов: **top** и **htop** — обязательно ознакомьтесь с ними!

Утилиты top и htop

Консольная утилита **top** служит для динамического просмотра списка процессов. Утилита каждые несколько секунд выдает топ-20 процессов, по умолчанию отсортированных по потреблению

процессорного времени. Можно сортировать список процессов по объему занимаемой памяти, утилизации процессора, пользователям или номерам процессов.

```
Файл  Правка  Вид  Закладки  Настройка  Справка
top - 12:15:46 up 1:31, 1 user, load average: 0,02, 0,02, 0,05
Tasks: 110 total, 1 running, 109 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0,3 us, 0,0 sy, 0,0 ni, 99,3 id, 0,0 wa, 0,0 hi, 0,3 si, 0,0 st
КиБ Mem: 3184748 total, 352104 used, 2832644 free, 40428 buffers
КиБ Swap: 2055164 total, 0 used, 2055164 free. 169860 cached Mem

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
  49 root        20   0     0     0     0  S   0,3   0,0   0:04.01 kworker/0:2
 218 oga         20   0   6684  1400 1052  R   0,3   0,0   0:00.03 top
   1 root        20   0   4540  2584 1440  S   0,0   0,1   0:01.89 init
   2 root        20   0     0     0     0  S   0,0   0,0   0:00.00 kthreadd
   3 root        20   0     0     0     0  S   0,0   0,0   0:00.43 ksoftirqd/0
   4 root        20   0     0     0     0  S   0,0   0,0   0:00.00 kworker/0:0
   5 root         0 -20     0     0     0  S   0,0   0,0   0:00.00 kworker/0:0H
   7 root        20   0     0     0     0  S   0,0   0,0   0:00.43 rcu_sched
   8 root        20   0     0     0     0  S   0,0   0,0   0:00.00 rcu_bh
   9 root        rt    0     0     0     0  S   0,0   0,0   0:00.00 migration/0
  10 root        rt    0     0     0     0  S   0,0   0,0   0:00.10 watchdog/0
  11 root         0 -20     0     0     0  S   0,0   0,0   0:00.00 khelper
  12 root        20   0     0     0     0  S   0,0   0,0   0:00.00 kdevtmpfs
  13 root         0 -20     0     0     0  S   0,0   0,0   0:00.00 netns
  14 root         0 -20     0     0     0  S   0,0   0,0   0:00.00 writeback
  15 root         0 -20     0     0     0  S   0,0   0,0   0:00.00 kintegrityd
  16 root         0 -20     0     0     0  S   0,0   0,0   0:00.00 bioset
  17 root         0 -20     0     0     0  S   0,0   0,0   0:00.03 kworker/u3:0
  18 root         0 -20     0     0     0  S   0,0   0,0   0:00.00 kblockd
  19 root         0 -20     0     0     0  S   0,0   0,0   0:00.00 ata_sff
```

top имеет горячие клавиши:

- **h** — вызов справки;
- **k** — ввести PID для команды kill;
- **r** — указать PID для изменения параметра **nice**. Влияет на приоритет процесса (-20 — максимальный приоритет, 19 — минимальный приоритет, 0 — по умолчанию). Обычный пользователь может только уменьшать приоритет своих процессов, назначая им **nice** от 0 до 19;
- **q** — завершить top.

Существует более продвинутая консольная утилита для мониторинга и управления процессами — **htop**. Она не установлена по умолчанию, но доступна из репозитория Ubuntu. Перечислим ее возможности в сравнении с **top**:

- позволяет выполнять горизонтальный и вертикальный скроллинг — и видеть все процессы и параметры командной строки;
- **htop** запускается быстрее top, которая предварительно собирает данные, прежде чем показать список процессов;

- в htop не надо вводить PID для kill, как в top — все через скроллинг и функциональные клавиши;
- также нет необходимости вводить PID и значение nice;
- в htop поддерживается мышь;
- top идет в системе по умолчанию и более распространена в мире UNIX.

```

Файл  Правка  Вид  Закладки  Настройка  Справка

CPU[ | 1.3%]      Tasks: 67, 72 thr; 1 running
Mem[||||| 138/3110MB]  Load average: 0.00 0.01 0.05
Swp[ 0/2006MB]    Uptime: 01:17:48

PID USER      PRI  NI  VIRT   RES   SHR  S  CPU% MEM%   TIME+  Command
---
2104 oga        20   0   6960   2048  1344  R  0.7  0.1   0:01.11 htop
1095 lightdm   20   0   241M  22500 17748  S  0.0  0.7   0:05.00 /usr/sbin/unity-gree
652 avahi     20   0   3604   1664  1388  S  0.0  0.1   0:00.69 avahi-daemon: runnin
968 root      20   0   9596   1004   708  S  0.0  0.0   0:01.06 /usr/sbin/VBoxServic
965 root      20   0   9596   1004   708  S  0.0  0.0   0:02.28 /usr/sbin/VBoxServic
972 root      20   0   9596   1004   708  S  0.0  0.0   0:00.82 /usr/sbin/VBoxServic
534 messagebu 20   0   4908   1972  1056  S  0.0  0.1   0:01.37 dbus-daemon --system
1259 lightdm   20   0  39400   8148  3468  S  0.0  0.3   0:00.63 /usr/lib/i386-linux-
1 root      20   0   4540   2584  1440  S  0.0  0.1   0:01.89 /sbin/init
262 root      20   0   3008    616   460  S  0.0  0.0   0:00.14 upstart-udev-bridge
281 root      20   0  12352   1680   992  S  0.0  0.1   0:00.05 /lib/systemd/systemd
412 root      20   0   2872    592   396  S  0.0  0.0   0:00.08 upstart-socket-bridg
676 root      20   0  37376   7944  3028  S  0.0  0.2   0:00.00 /usr/sbin/ModemManag
692 root      20   0  37376   7944  3028  S  0.0  0.2   0:00.00 /usr/sbin/ModemManag
569 root      20   0  37376   7944  3028  S  0.0  0.2   0:00.02 /usr/sbin/ModemManag
593 root      20   0   4212   1772  1464  S  0.0  0.1   0:00.01 /lib/systemd/systemd
596 root      20   0   4880   1680  1476  S  0.0  0.1   0:00.00 /usr/sbin/bluetoothd
600 syslog    20   0  30476   1180   860  S  0.0  0.0   0:00.01 rsyslogd
601 syslog    20   0  30476   1180   860  S  0.0  0.0   0:00.00 rsyslogd
602 syslog    20   0  30476   1180   860  S  0.0  0.0   0:00.01 rsyslogd
597 syslog    20   0  30476   1180   860  S  0.0  0.0   0:00.04 rsyslogd
653 avahi     20   0   3476    432   216  S  0.0  0.0   0:00.00 avahi-daemon: chroot
671 root      20   0   2884    600   360  S  0.0  0.0   0:00.04 upstart-file-bridge
688 root      20   0  55028  10484  5320  S  0.0  0.3   0:00.00 NetworkManager
704 root      20   0  55028  10484  5320  S  0.0  0.3   0:00.17 NetworkManager

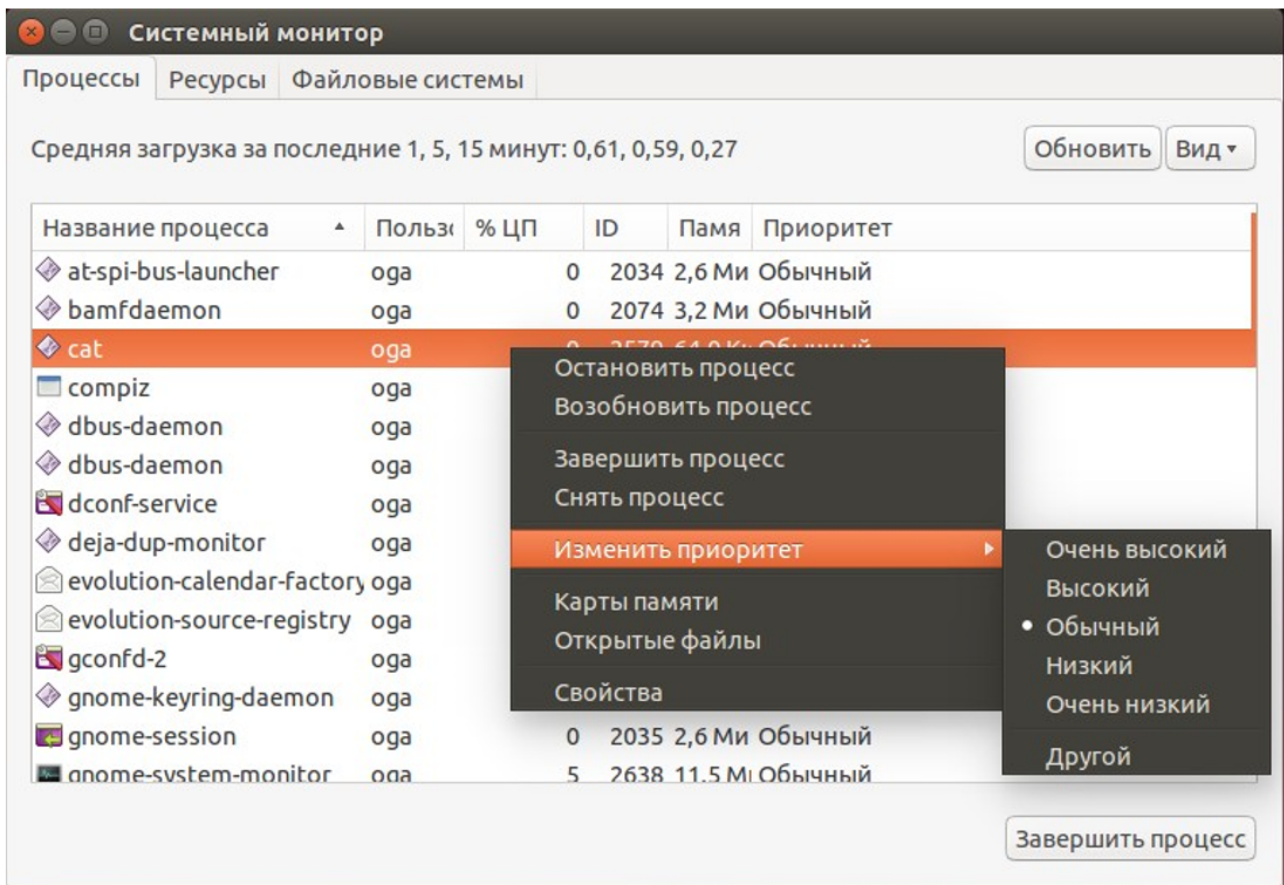
F1 Help  F2 Setup  F3 Search  F4 Filter  F5 Tree  F6 SortBy  F7 Nice -  F8 Nice +  F9 Kill  F10 Quit

192.168.9.110 : oga

```

Системный монитор в X-Window

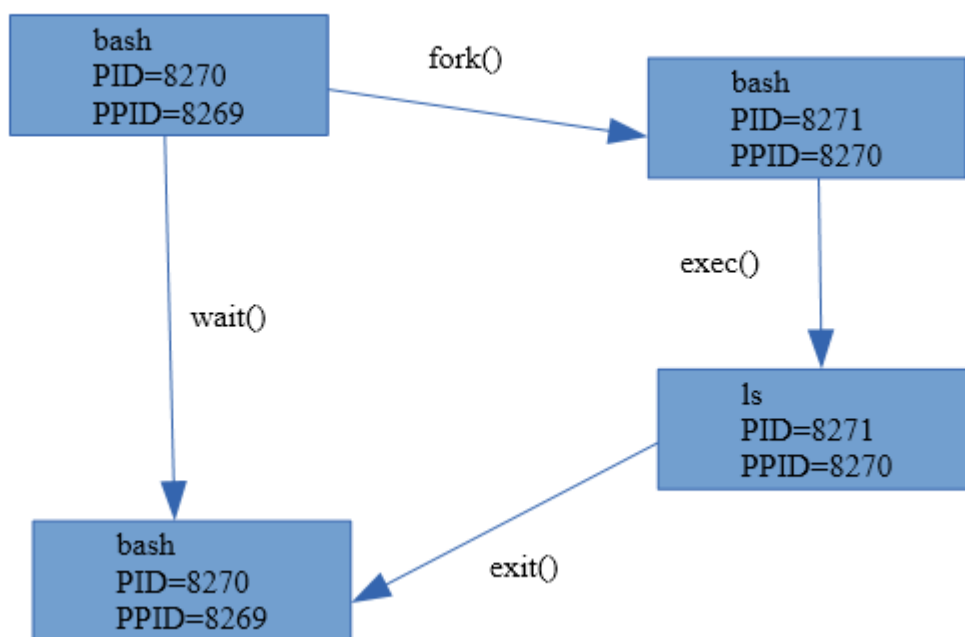
Для мониторинга ресурсов и управления процессами в графической среде X-Window служит программа «Системный монитор». Для запуска найдем ее в главном меню, набрав в поиске программ System Monitor. Управление процессами интуитивно понятно. Меню «Вид» позволяет выбрать процессы пользователя или посмотреть все запущенные процессы.



Порядок запуска процесса

Рассмотрим порядок запуска процесса.

Работая в оболочке, мы запустили команду `ls`:



На первом этапе оболочка выполняет системный вызов `fork()`, в результате которого происходит ее клонирование: создается полная копия процесса оболочки, включая копию адресного пространства и контекста. Между клонами есть отличие: в процессе-родителе `fork()` возвращает PID потомка, а в потомке код возврата — 0. Каждый из клонов начинает выполнение с момента вызова `fork`, но уже самостоятельно и с учетом вышесказанного. Процесс, выяснив, что он потомок (так как код возврата 0), с помощью системного вызова `exec` загружает в свое адресное пространство исполняемый код запускаемой программы, в нашем случае `ls`, и далее ее выполняет.

В это же время родительский процесс ожидает завершения дочернего с помощью системного вызова `wait()`. Если мы запустили команду с `&` в конце, ожидания не будет.

Дочерний процесс завершается функцией `exit()`. В случае успешного завершения `exit` возвращает 0, если произошла ошибка — код ошибки, отличный от нуля. После этого ядро освобождает ресурсы, которые занимал потомок, и передает код его завершения родителю в качестве кода возврата для `wait()`.

Иногда бывает так, что процесс-родитель прекращает существование, не успев дождаться завершения `wait()`, то есть раньше завершения потомка. Процесс, у которого процесс-родитель завершился раньше, называют сиротой. Процессы-сироты усыновляет процесс `init`, который теперь считается для них родителем.

Утилита `kill` и сигналы процессам

Представим, что программа зависла. Нажимаем Alt-F2 (или Ctrl-Alt-F2, если работали в X-сервер), смотрим id процесса.

```
$ ps ax | grep hello
```

Процессу можно направить сигнал.

Например, сигнал 9 — принудительно снять.

```
$ kill -9 4242
```

В Linux существуют следующие сигналы:

- 1 (SIGHUP) — информирует программу о потере связи с терминалом. Ситуация часто возникала в прошлом, в режиме терминального доступа. В настоящее время может применяться для двух целей:
 - информирования дочернего процесса о завершении родительского. При завершении родительского процесса ядро направит сигнал 1 дочерним процессам.

Если приложение запущено в фоновом режиме, можно указать `&` в конце команды или `fg` при завершении консоли. Поступит сигнал SIGHUP.

```
$ someprogram&
```

- либо:

```
$ someprogram  
  
^Z  
  
$ fg
```

Этому можно воспрепятствовать с указанием **nohup**:

```
$ nohup someprogram&
```

Очевидно, что по умолчанию сигнал приводит к завершению программы. Но демоны перехватывают данный сигнал и используют для перечитывания файлов конфигурации.

- 2 (SIGINT) формируется при нажатии Ctrl-C. Завершает программу, но может перехватываться или блокироваться (например, в программах, в которых Ctrl-C используются для операции копирования, или где в принципе не нужен такой вариант остановки).
- 8 (SIGFPE) — ошибка операции с целочисленной арифметикой (переполнение, деление на ноль). Сохраняется дамп памяти.
- 9 (SIGKILL) — безусловное завершение программы. Сигнал не может быть перехвачен программой, поэтому позволяет ее остановить в любом случае (но не позволит снять процесс-зомби).
- 11 (SIGSEGV) — формируется при попытке программы обратиться к не принадлежащей ей области памяти. Обычно выводится сообщение «Segmentation fault» и сохраняется дамп памяти. Как правило, такое случается в результате ошибок программиста при работе с указателями.
- 15 (SIGTERM) — вежливая просьба программе завершить работу. Программа может сохранить данные и т. д.
- Есть и другие варианты.

Для отправки сигнала из операционной системы можно использовать команду **kill**.

```
$ kill -15 4242
```

Перенаправление потоков ввода-вывода

Познакомимся с подходом UNIX-Way. Один из его принципов: каждая программа хорошо отлажена и решает только одну задачу, но сами программы-утилиты можно комбинировать, что очень часто применяется. Сейчас мы разберем, каким образом это работает.

При запуске каждая программа автоматически открывает три специальных файла — их еще называют потоками ввода-вывода. Их различают по номерам файловых дескрипторов (указателей на файл):

- 0 — стандартный ввод (`stdin`). По умолчанию принимается с клавиатуры терминала;
- 1 — стандартный вывод (`stdout`). По умолчанию — на экран терминала;
- 2 — стандартная ошибка (`stderr`). По умолчанию — на экран терминала.

Поток стандартного вывода можно перенаправить в файл (в том числе специальный, такой как принтер или файл «в никуда» `/dev/null`). Стандартный ввод программе можно назначить не с клавиатуры, а из файла. Опишем все варианты перенаправления для каждого потока:

Для `stdout` от программы `cmd` можно использовать символы перенаправления:

- `cmd > file` перенаправить в файл с перезаписью (при каждой записи предыдущее содержимое файла будет утрачено),
- `cmd >>file` в файл с добавлением (новые записи будут добавлены в конец файла),
- `cmd1 | cmd2` — конвейер. Запускаются `cmd1` и `cmd2`, при этом `stdout` от `cmd1` перенаправляется на `stdin` программы `cmd2`,
- `cmd | tee file` — перенаправить копию потока в файл. После `tee` с потоком `stdout` можно выполнять дополнительные операции,
- `cmd | sudo tee file` — вариант, если в `file` имеет возможность записать только `root`.

Для `stdin` программы `cmd` можно использовать символы перенаправления:

- `cmd < file` перенаправить ввод из файла программе.

Для `stderr` от программы `cmd` можно использовать символы перенаправления:

- `cmd 2> file` перенаправить в файл с перезаписью;
- `cmd 2>>file` в файл с добавлением;
- `cmd1 2>&1` — присоединить к `stdout`;
- после операций с потоком `stderr` можно выполнять дальнейшее перенаправление для `stdout`.

Например, с помощью команды `tee` мы продублируем поток `stdout` на терминал и перенаправим поток в три файла: `f1`, `f2`, `f3`:

```
cmd 2> /dev/null | tee f1 f2 f3
```

Перенаправить поток ошибок в `/dev/null` (спрятать поток), поток стандартного вывода — на вход программе `tee`, которая дублирует его в файлы `f1`, `f2`, `f3`:

```
cmd < f1 | cmd2 2>&1 > f2
```

Направить программе `cmd` ввод из файла `f1`. Затем стандартный вывод `cmd` направить на вход `cmd2`. Объединить поток ошибок от `cmd2` со стандартным выводом от `cmd2` и все перенаправить в `f2`.

Создать файл.

- **>file1** — самый простой способ создать пустой файл;
- **cat >file1** ввести файл с консоли (завершение — Ctrl-Z).

Установка программ в Ubuntu

Самые базовые возможности установки мы разобрали на прошлом уроке. Теперь подробнее.

Большинство пользователей знают, как устанавливаются программы в Windows. Многие новички в Linux страдают от того, что опираются на эти навыки. В Linux почти все делается по-другому. Первое, что надо осознать: в Linux всегда существует несколько способов решить одну и ту же проблему. Но это не значит, что какой-то из этих способов не будет лучше других. Поэтому рассмотрим разные пути установки программ — от предпочтительного к наименее рекомендованному к использованию.

Ключевые понятия

Репозиторий — хранилище пакетов. Обычно он располагается на сервере онлайн, хотя ничто не мешает создать локальный репозиторий или репозиторий на DVD. Под пакетом понимают файл **.deb** (в Ubuntu), который содержит в упакованном виде необходимые программные файлы, инсталляционные скрипты и разного рода дополнительную информацию. Его можно условно сравнить с файлами **.msi** в Windows.

Часто для работы программ требуются дополнительные библиотеки — без них пакеты не установятся. Это называется зависимостями. В Windows тоже можно заметить подобное. Например, некоторые игры требуют присутствия в системе определенной версии **DirectX**.

В Linux эта практика более распространена. Такой подход позволяет существенно уменьшить трафик, а также сэкономить место на диске, но иногда усложняет сам процесс установки. Также отметим, что **.deb**-пакеты присутствуют в ОС GNU/Linux, основанных на Debian и Ubuntu. В других системах — другие форматы пакетов и утилиты, например в RedHat и Centos — **.rpm**. Но в целом идеи во многом похожие, и, например, в Centos вместо выполнения команды **sudo apt install mc** вы будете выполнять команду **sudo yum install mc**.

Основной способ установки программ. Использование официальных репозиториев

Самый простой способ установить новую программу в Ubuntu — выбрать пункт меню **Программы > Установить / Удалить**. Дальше выбрать необходимую программу в каталоге, поставить галочку и нажать **Применить**. Все остальное будет сделано автоматически. Система сама загрузит необходимые пакеты из интернета, установит их и добавит новый пункт в меню.

Если вы хотите иметь больше возможностей влиять на процесс, используйте более продвинутый инструмент — **Synaptic**. Выберите **Система > Администрирование > Менеджер пакетов Synaptic**. С ним вы можете устанавливать программы частично, если вам не нужна документация или еще что-то.

Можно использовать и консольные инструменты. Если хотите установить программу для записи дисков Brasero, наберите команду **sudo apt-get install brasero** — и она установится.

У такого метода установки программ есть масса преимуществ: это удобно, вы устанавливаете уже протестированные программы, которые гарантированно будут работать на вашей системе,

зависимости между пакетами будут решаться автоматически, вы сразу узнаете о появлении в репозитории новых версий установленных программ.

Использование сторонних репозиториев

В официальных репозиториях программ очень много, но при их использовании могут возникать проблемы. Например, некоторые программы могут присутствовать в уже устаревших версиях, а некоторых просто может не быть. При этом ряд проектов имеет собственные репозитории: Wine, Elisa и другие. В таком случае на официальном сайте нужной вам программы должно быть описано, как такие репозитории подключить. После подключения репозитория процесс установки не отличается от описанного выше.

Непосредственная установка из deb-пакетов

Всегда можно загрузить готовые deb-пакеты с помощью браузера. Существуют порталы с выложенными deb-пакетами программ, такие как **GetDeb.net**. Также можно загружать файлы напрямую с официального сайта нужной вам программы. Скачанные пакеты легко устанавливаются двойным кликом мышки. Минус такого подхода — менеджер обновлений не будет отслеживать появление новых версий установленной программы.

Если вам попался пакет **rpm** (для систем типа Fedora, Mandriva, SUSE и т. д.), его можно сконвертировать в **deb** с помощью утилиты **alien** (устанавливается отдельно). **Однако** делать это рекомендуется только в исключительных случаях, когда пакетов deb в доступе нет (например, драйверы принтеров Canon на официальном сайте выложены исключительно в **rpm**).

Установка программ с собственным инсталлятором

Иногда программы могут распространяться с собственным инсталлятором. Это ничем не отличается от ситуации в Windows. Только здесь, распаковав **tar.gz**-архив с дистрибутивом программы, вы вместо **setup.exe** увидите что-то вроде **INSTALL.sh**. Если этот файл не захочет запускаться — вероятно, ему не даны права на запуск. Иногда дистрибутив программы распространяется в виде самораспаковывающегося архива. В таком случае это будет просто один-единственный файл **.sh**, который нужно запустить. Дальше вы получите мастер, где нужно будет ответить на ряд вопросов — так же, как это делается в Windows. Так устанавливаются официальные драйверы nVidia, ATI, среда разработчика NetBeans.

Установка программ, которые не нуждаются в инсталляции

Есть программы, которые распространяются в виде обычного архива **tar.gz** (их часто называют «тарболлы»), который достаточно куда-то распаковать. В Windows тоже есть такие программы (Portable). Устанавливать их можно куда угодно, но стандартное место — каталог **/opt**. Пункты на запуск в меню вам придется добавлять вручную: для этого нужно щелкнуть правой кнопкой по заголовку меню «Программы» и выбрать «Правка меню».

Установка программ из исходных текстов

Обычно следует избегать этого способа установки программ. Но иногда такой способ установки может оказаться единственно доступным. Основное, что понадобится, — это средства для компиляции. Сначала нужно установить пакет **build-essential**. Потом распаковать архив с кодами программы во временную папку. Затем найти файл README или INSTALL, прочитать его и выполнить то, что в нем

написано. Чаще установка программ таким способом ограничивается последовательным выполнением следующих команд:

```
./configure make  
  
sudo make install
```

В некоторых случаях последовательность команд для установки может отличаться. После выполнения скрипта **./configure** вы можете получить сообщение о том, что в системе не установлено нужных библиотек для компиляции программы. В таком случае нужно будет установить их и повторить процесс. Обычно компиляция занимает время — продолжительность напрямую зависит от мощности компьютера.

Репозитории в UBUNTU

Репозитории — это специальные сервера-хранилища файлов. Их также можно назвать источниками приложений. Пользовательские компьютеры подключаются к репозиториям по сети или через интернет и при помощи специальных утилит (например, Synaptic) позволяют увидеть, какие пакеты установлены или доступны для установки. Большинство утилит поддерживают простой поиск по ключевым словам и способны разбивать группы пакетов по категориям.

Связка «репозиторий — утилита» позволяет использовать централизованный метод установки и удаления программ, а также предоставляет удобный способ выкладывания обновлений.

В Ubuntu уже подключены необходимые репозитории, однако никто не запрещает использовать сторонние репозитории.

В репозиториях Ubuntu содержится достаточно разнообразного ПО, и зачастую вы можете найти там необходимую программу. Только если вы не нашли нужную программу в репозиториях, стоит попробовать другие пути установки.

Версии ПО, доступные через репозитории, не всегда свежие. Но они, как правило, лучше интегрированы в Ubuntu и более стабильны.

Структура репозитория Ubuntu

В Ubuntu все программное обеспечение делится на четыре секции — компоненты. Они отражают разницу в лицензии и уровне доступной поддержки. Пакеты распределяются по компонентам таким образом:

- **Main** — свободное ПО, официально поддерживаемое компанией Canonical;
- **Restricted** — проприетарное ПО (в основном — драйверы устройств), официально поддерживаемое компанией Canonical;
- **Universe** — свободное ПО, официально не поддерживаемое компанией Canonical, но обеспеченное поддержкой сообщества пользователей;
- **Multiverse** — проприетарное ПО, не поддерживаемое компанией Canonical.

Четыре основных репозитория Ubuntu:

- **\$release** — пакеты на момент выхода релиза;

- **\$releasesecurity** — пакеты критических обновлений безопасности;
- **\$releaseupdates** — пакеты обновления системы (более поздние версии ПО, вышедшие уже после релиза);
- **\$releasebackports** — бэкпорты более новых версий ПО, которое доступно только в нестабильных версиях Ubuntu;
- **partner** — репозиторий, содержащий ПО компаний-партнеров Canonical.

Кроме официальных, существует множество репозиториев от авторов программ и от тех, кто не поленился собрать из исходников пакет и поделиться им.

Подключение репозитория

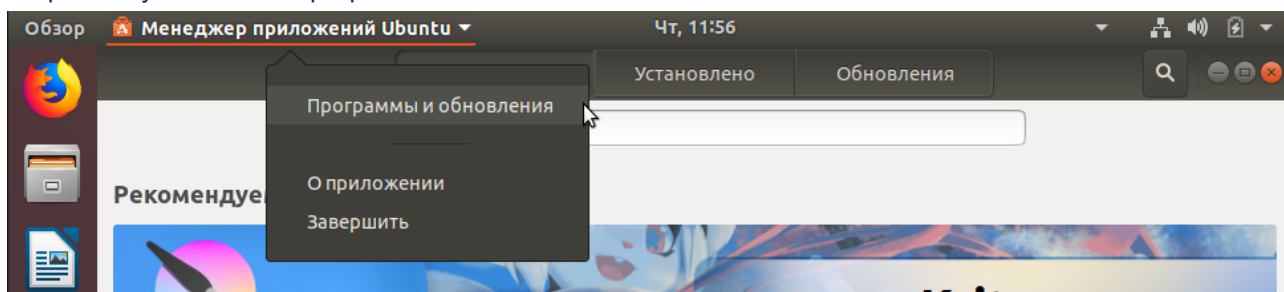
Есть много сторонних репозиториев, подключив которые вы получите доступ к дополнительному ПО. Сделать это можно как при помощи графического интерфейса, так и в консоли.

Некоторые репозитории помимо нужных пакетов могут содержать экспериментальные сборки различного системного ПО, в том числе и ядер Linux. Поскольку версия этих экспериментальных пакетов, как правило, выше, чем установленная, менеджер обновлений может попытаться «обновить» систему с этих репозиториев, что может повредить ее. Поэтому внимательно читайте описание подключаемого репозитория и информацию в менеджере обновлений.

Подключение при помощи графического интерфейса

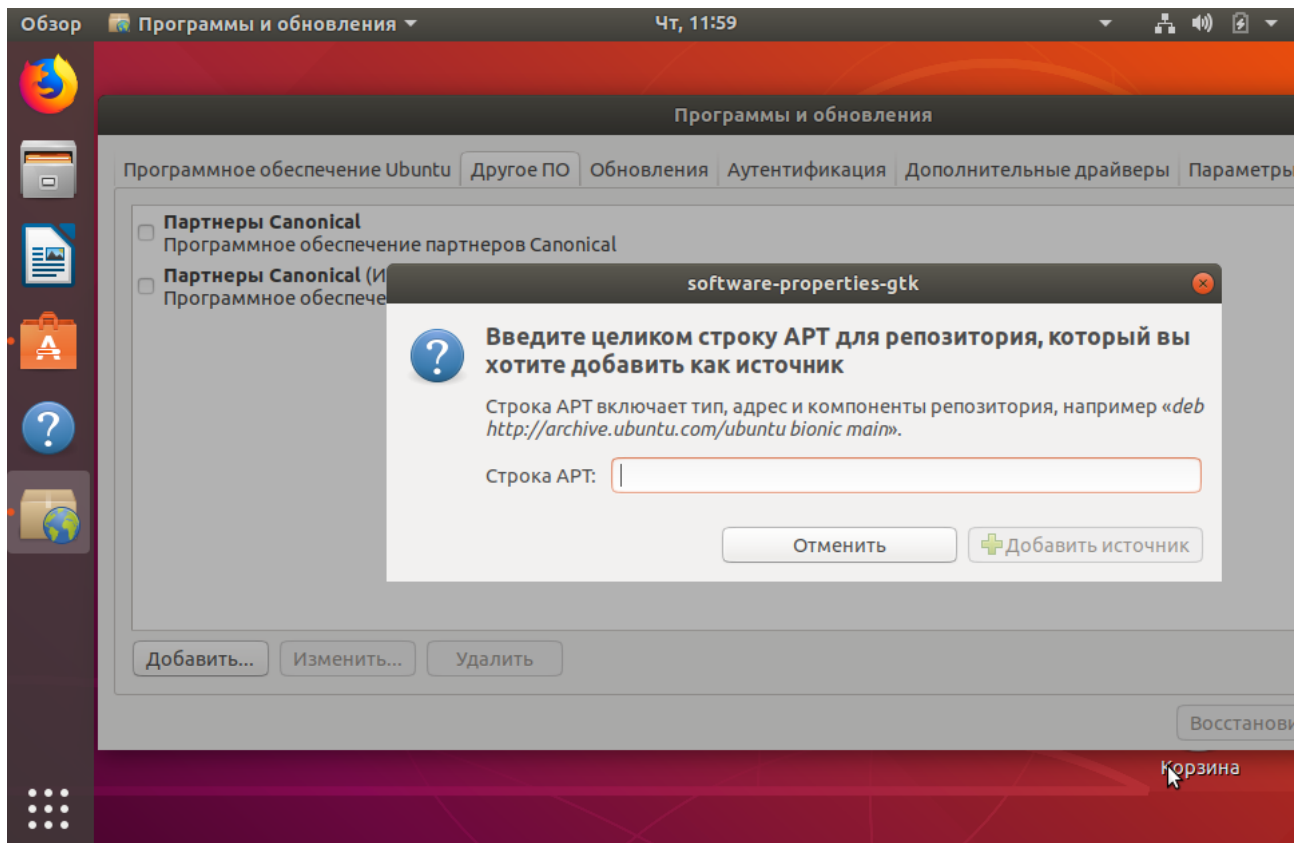
Для подключения репозитория выполните следующие шаги:

1. Откройте менеджер приложений Ubuntu.
2. Откройте пункт меню «Программы и обновления...»:



3. В появившемся окне выберите вкладку «Другое ПО», нажмите кнопку «Добавить».

4. В появившемся окне заполните поле «Строка APT:» и нажмите кнопку «Добавить источник».



Так как был подключен новый источник программного обеспечения, необходимо обновить информацию о пакетах. Появится окно с предложением это сделать. Нажмите «Обновить».

После обновления информации о пакетах окно «Источники приложений» закроется, и, скорее всего, вы получите ошибку о неподписанном источнике приложений. Тем не менее вы сможете устанавливать пакеты, содержащиеся в подключенном репозитории, стандартными средствами. Для устранения ошибки неподписанного репозитория смотрите пункт про защиту репозитория ниже.

При помощи консоли (рекомендуемый способ)

В современных версиях Ubuntu добавлять репозиторий можно одной командой. Вот пример для ppa-репозитория:

```
sudo add-apt-repository ppa:cpick/hub
```

Системный список репозитория содержится в файле `/etc/apt/sources.list`. Чтобы добавить репозиторий, отредактируйте файл так:

```
sudo nano /etc/apt/sources.list
```

Добавьте туда APT-строку. Чем выше (ближе к началу файла) стоит строка, тем больший приоритет получит добавленный репозиторий.

Далее следует обновить список пакетов. Для этого выполните:

```
sudo apt-get update
```

Теперь вы можете устанавливать пакеты из нового репозитория:

```
sudo apt install hub
```

Устройство репозитория

Пакет (например, *.deb-файл) размещается на общедоступном интернет-ресурсе (например, archive.ubuntu.com). Затем информация о пакете заносится в файл **Packages**, который, в свою очередь, для удобства пакуется в **Packages.gz**.

Файлов **Packages.gz** может быть несколько (например, по одному для каждой архитектуры). Файл **Release** содержит описание репозитория в целом и ссылки на различные **Packages.gz**.

Общая схема работы выглядит примерно так:

1. Пользовательский компьютер подключается к репозиторию, и, при наличии защиты, проверяет его истинность (см. главу «Защита репозитория»).
2. Читает файл **Release**, находит и скачивает необходимые **Packages.gz**.
3. На основе скачанных **Packages.gz** обновляет локальную базу данных пакетов.
4. Теперь пользовательский компьютер знает, где находится тот или иной пакет, и при необходимости легко может его скачать и установить.

Защита репозитория

Поскольку репозитории по большей части расположены в интернете, существует вероятность подмены их злоумышленником на свои, содержащие модифицированные пакеты. Пользователь может установить модифицированный пакет и тем самым поставить безопасность системы под угрозу. Многие репозитории имеют защиту от подмены. Такая защита реализована при помощи сверки цифровых подписей репозитория и клиента. Когда репозиторий имеет цифровую подпись, а пользовательский компьютер содержит открытый ключ для него, репозиторий считается доверенным.

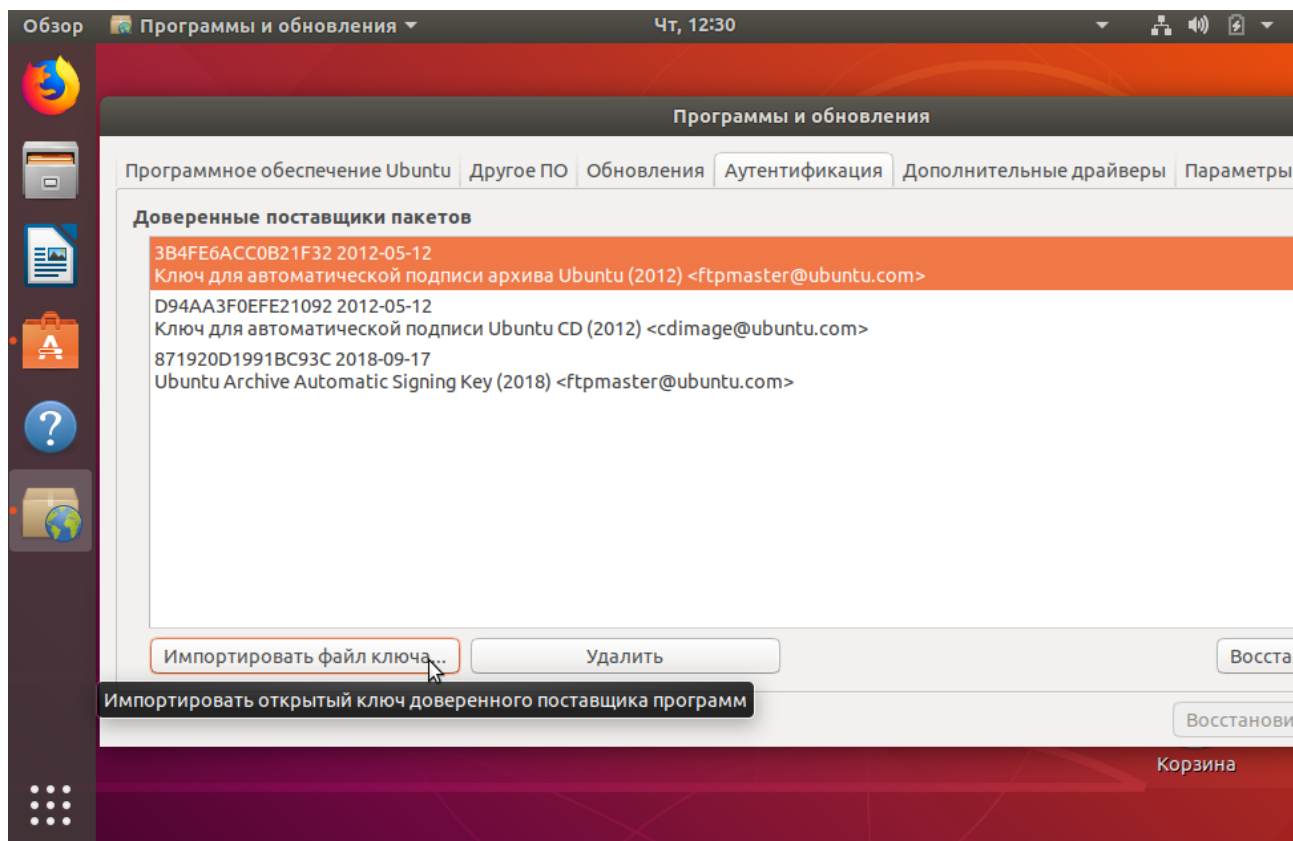
В Ubuntu по умолчанию доверенными являются репозитории на установочных дисках и основные интернет-репозитории — **archive.ubuntu.com**. При наличии на пользовательском компьютере нескольких подключенных репозитория предпочтение отдается доверенным.

При подключении репозитория, защищенного цифровой подписью, вам нужно скачать открытый ключ — обычно с ресурса, рассказывающего про этот репозиторий, или с сервера ключей (предпочтительно). Затем этот ключ надо добавить в систему. Иногда для скачивания предоставляется доступный пакет, который при установке сам прописывает ключ репозитория. Если вы скачиваете ключ с сайта репозитория, вы получаете обычный файл с расширением **.key**, **.gpg** или другим. Добавить его в систему можно так:

```
sudo apt-key add repo.key
```

repo.key — полученный ключ репозитория.

Или при помощи графического интерфейса запустите «Источники приложений» (Система — Администрирование — Источники приложений), перейдите на вкладку «Аутентификация» и нажмите на кнопку «Импортировать файл ключа...». Откроется диалог выбора файла — выберите файл ключа и нажмите ОК.



Но гораздо более предпочтительно добавление ключа со специального защищенного сервера. Обычно, когда заходит речь о ключе, дается его непонятный с первого взгляда идентификатор вида 123ABCDEF456 (строка из произвольных цифр и букв латинского алфавита в верхнем регистре). Это уникальное имя ключа. Иногда ключ описывается строкой вида 1024R/123ABCD — тогда идентификатором является часть после слеша. Ключи преимущественно хранятся на специальных серверах, откуда любой может их получить. Ключи для репозитория Ubuntu принято хранить на **keyserver.ubuntu.com**. Для получения и импортирования в систему ключа с сервера необходимо выполнить команду:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com recvkeys 12345678
```

Вместо **keyserver.ubuntu.com** можно подставить адрес другого сервера ключей, а вместо **12345678** написать идентификатор нужного вам ключа.

Чтобы разом попытаться импортировать все недостающие ключи репозитория, выполните в консоли:

```
sudo apt-key adv recvkeys --keyserver keyserver.ubuntu.com `sudo aptitude update 2>&1 | grep o '[09AZ]\{16\}$' | xargs`
```

Здесь присутствует регулярное выражение. Ряд часто встречаемых регулярных выражений и команд с их использованием можно найти в интернете, но еще лучше уметь их писать — мы научимся этому на следующем уроке.

Логи, просмотр логов

После установки программы или в процессе ее работы могут возникнуть проблемы и ошибки, которые требуется просмотреть и проанализировать.

Для хранения логов предусмотрена директория `/var/log`.

При этом имеются как системные логи, так и логи для отдельных программ.

Общесистемные логи, как правило, хранятся в файле `/var/log/syslog` (или в `/var/log/messages`, например в Centos).

Сообщения ядра хранятся в `/var/log/dmesg` (может понадобиться при добавлении драйверов-модулей ядра).

Записи в журналах обычно содержат метку времени, имя хоста, на котором выполняется описываемый процесс, и имя процесса.

Просматривать журналы можно при помощи программы постраничного вывода, например **less**. Искать определенные записи (например, сообщения ядра от определенного демона) можно при помощи команды **grep**:

```
# less /var/log/messages
```

или

```
# grep "ppp" /var/log/messages | tail
```

Посмотреть последние 25 строк файла можно командой **tail**:

```
$ tail -25 /var/log/syslog
```

Без указания ключа **tail** выдаст последние 10 строк.

Если нужно выдавать последние строки файла по мере их появления, используйте ключ **-f**:

```
$ tail -f /var/log/syslog
```

Чтобы прервать работу **tail -f**, используйте комбинацию **Ctrl-C**. Есть команда, аналогичная **tail**, только для начала файла — **head**. Без параметров выдает первые 10 строк указанного файла:

```
$ head filename
```

Встроенная документация. Команды `man`, `apropos`

На прошлом занятии мы уже касались поиска информации о нужной программе. Разберем более подробно возможности работы с документацией.

В Unix существует встроенная система документации, обратившись к которой всегда можно найти название нужной утилиты, описание параметра команды или пример ее использования. Например, если требуется подробное описание команды `cp`, используем команду `man` (от MANUAL).

```
user@user-virtual-machine:~$ man cp
```

Справка по командам обычно занимает несколько страниц, поэтому `man` использует для просмотра команду `less`. Для перехода к следующей странице используйте клавишу <PgDn>, обратно — <PgUp>. Если эти клавиши не срабатывают, используйте для перехода пробел и клавишу `u` соответственно. Для выхода из `man` служит клавиша `q`. Есть еще пара полезных клавиш: `g` — переход в начало, `G` — в конец документа. Для поиска вперед используйте *Искать*, для обратного поиска — *Что искать*, повторить поиск в том же направлении — `n`.

Теперь пара слов о стандартных разделах справочной страницы.

NAME — краткое описание команды.

Раздел **SYNOPSIS** описывает различные варианты синтаксиса. Здесь могут использоваться специальные обозначения. Например, если опции или параметры заключены в квадратные скобки, это означает, что они не обязательны для использования. Три точки после параметра говорят о том, что он может повторяться многократно, как в случае, когда несколько файлов копируются в один каталог назначения.

В разделе **DESCRIPTION** описываются опции программы (иногда их называют ключами). Бывает, что в описании через запятую перечислены две опции, например `-v, --verbose`. В таких случаях мы имеем дело с разными стилями одной опции. `-v` — классический вариант, описанный в стандарте **POSIX**. `--verbose` — опция в стиле GNU, новый вариант. Можно использовать то, что больше нравится.

Еще один полезный раздел обычно находится в конце справочной страницы — **SEE ALSO**. Как правило, здесь приводят ссылки на другие утилиты с похожей функциональностью.

Кроме `man` есть еще несколько полезных справочных утилит. Если мы не знаем точного имени утилиты, можно попробовать найти ее с помощью `apropos`, которая выполнит поиск по ключевым словам в описании утилит.

Найдем, как называется утилита, которая меняет пароль пользователя:

```
user@user-virtual-machine:~$ apropos password
apg (1)          - generates several random passwords
chage (1)        - change user password expiry information
chgpaswd (8)     - update group passwords in batch mode
chpaswd (8)     - update passwords in batch mode
cpgr (8)        - copy with locking the given file to the password
or g..
cppw (8)        - copy with locking the given file to the password
or g..
cracklib-check (8) - Check passwords using libcrack2
create-cracklib-dict (8) - Check passwords using libcrack2
crypt (3)       - password and data encryption [...]
```

Мы получили нужную информацию, но слишком много вариантов — на три страницы.

Попробуем уточнить поиск, выбрав все строки, где встречается слово **change**. Используем утилиту **grep**, которая ищет по заданной строке в выводе от предыдущей программы. Делается это так:

```
user@user-virtual-machine:~$ apropos password | grep change
chage (1)          - change user password expiry information
lppaswd (1)        - add, change, or delete digest passwords.

passwd (1)         - change user password
smbpasswd (8)     - change a user's SMB password
```

Вот она, **passwd** — то, что нужно. Обратите внимание на число в скобках после имени команды. Это глава или раздел справочного руководства **man**, в которых может встречаться описание утилиты. Разделов много, приведем самые полезные:

- 1 — команды пользователя,
- 2 — системные вызовы ядра (используется программистами),
- 5 — форматы файлов,
- 8 — команды администрирования.

Эти номера можно использовать в командах **man** и **apropos**. Например, **passwd** — не только команда, но и имя системного файла. Если нам интересен формат этого файла, надо набрать:

```
man 5 passwd
```

Для **apropos** полезна опция **-s**. Например, если мы ищем программы для работы с паролями, набираем:

```
user@user-virtual-machine:~$ apropos -s 1 passwd
gpasswd (1)          - управление /etc/group и
/etc/gshadow
passwd (1)          - изменяет пароль пользователя
grub-mkpasswd-pbkdf2 (1) - generate hashed password for
GRUB
lpasswd (1)         - add, change, or delete digest
passwords.
passwd (1ssl)       - compute password hashes
user@user-virtual-machine:~$
```

Описание **passwd** — по-русски. Иногда это может создать проблемы для поиска, поэтому есть еще полезная опция **-L**: она задает язык, на котором будет справка. Чтобы все искать и читать по-английски, задаем **apropos -s 1 -L C passwd**.

```
user@user-virtual-machine:~$ apropos -s 1 -L C passwd
gpasswd (1)          - administer /etc/group and
/etc/gshadow
grub-mkpasswd-pbkdf2 (1) - generate hashed password for
GRUB
lpasswd (1)         - add, change, or delete digest
passwords.
passwd (1)          - change user password
passwd (1ssl)       - compute password hashes
user@user-virtual-machine:~$
```

Еще одна полезная справочная программа — **whatis**. От **apropos** отличается тем, что ищет по именам команд:

```
user@user-virtual-machine:~$ whatis passwd
passwd (5)          - файл паролей
passwd (1)          - изменяет пароль
пользователя
passwd (1ssl)       - compute password hashes
```

Иногда в разделе справочной страницы SEE ALSO встречается ссылка на более полное описание утилиты в системе **info** и проводится команда для получения полной справки. Пример для **cp**:

```
info coreutils 'cp invocation'
```

Info — гипертекстовая система документации вроде текстового браузера, а команда в **man** содержит полную ссылку на информацию о вызове команды в **info**. Если запустить **info** без параметров, мы выйдем на главную страницу документации. Перемещение по страницам и поиск — как в **man**.

Строчки, которые начинаются со звездочек, — гиперссылки, по которым можно пойти дальше. Чтобы вернуться из ссылки на уровень выше, используйте клавишу **u** — как **Back** в браузере. Выход — по клавише **q**.

Архиваторы и компрессоры в Linux

Наверняка вы использовали в Windows программы-архиваторы (**rar** и **zip**), которые позволяют упаковать иерархию каталогов с файлами в один сжатый архивный файл. В Linux тоже есть версии этих архиваторов, которыми можно распаковать архивы, созданные пользователями Windows. Также есть графические программы-оболочки, облегчающие пользователям работу с архивами. Но сейчас мы поговорим о других программах из мира UNIX, которые появились задолго до своих Windows-собратьев. Чем они уникальны?

Во-первых, есть различие в терминах. Архиваторами в UNIX и Windows называют программы, немного отличающиеся по назначению. В UNIX архиваторы выполняют следующие задачи:

- создавать архивы как объединения заданных файлов в виде одного обычного файла или последовательности данных на внешнем носителе (диск, лента и т. д.);
- добавлять в архив или извлекать из него отдельные файлы или группы файлов.

В Windows архиваторы делают то же, что в UNIX, и дополнительно занимаются сжатием архивов. В UNIX сжатием занимаются отдельные утилиты — компрессоры. Такое разделение функций является характерным для философии Unix way: каждой задаче — свой продвинутый инструмент, плюс комбинация разных инструментов для решения новых задач.

Подход оказался оправданным:

- он позволяет использовать новые, более мощные компрессоры в сочетании со старыми архиваторами;
- и с помощью конвейеров добавлять в рабочую цепочку «архиватор — компрессор» другие программы. Например, для шифрования данных, что очень актуально для хранения архивов на популярных облачных сервисах.

Далее рассмотрим конкретные примеры программ для архивации и сжатия файлов.

В UNIX с давних времен существуют два стандартных архиватора: **tar** и **cpio**. В Linux также есть GNU-версии этих утилит. У них есть уникальные опции с длинными именами, но для совместимости они поддерживают стандартный набор ключей. Будем придерживаться стандарта, чтобы вы могли использовать утилиты в любой UNIX-системе.

Утилита tar (Tape ARchiver)

Судя по названию, предназначалась для создания архивов на магнитных лентах, но также прекрасно работает с архивами в виде файлов.

Создание архивов с tar

tar опции **arc_file file_or_dirname ...**

Значение опций:

- **c** — создать архив;

- **z** — сжимать файл архива с помощью **gzip** (см. ниже). Эта опция есть только в GNU-версии;
- **j** — сжимать файл архива с помощью **bzip2** (см. ниже). Эта опция есть только в GNU-версии;
- **v** — (**verbose**) — выдает в терминал имя добавляемого файла. Когда файлов много, обычно эту опцию не указывают и **tar** работает молча;
- **f** — после этой опции указывают имя архивного файла. Часто используют специальное имя «-». Оно означает, что поток данных архива надо выдать в стандартный выходной поток. В этом режиме обычно используют конвейерную обработку, когда данные архиватора принимаются компрессором для последующего сжатия. Если забыть указать файл после **-f**, **tar** будет пытаться писать на ленточное устройство по умолчанию;
- **file_or_dirname** — список архивируемых файлов или каталогов. Каталоги архивируются рекурсивно вместе со всем содержимым.

Пример:

```
tar cf foo.tar ~/foo
```

Просмотр оглавления архива

```
tar tvf arc_file
```

t — прочитает оглавление архива. Остальные опции — как и в случае создания архива. Пример:

```
tar tvf foo.tar
```

Извлечь файлы из архива

```
tar xvf arc_file [filename]
```

x — извлечь файл(ы). Если последний параметр не указывать, будут извлечены все файлы из архива в текущий каталог. Имена извлекаемых файлов надо указывать, так как они показаны в листинге оглавления архива (с тем же относительным путем).

Компрессоры

В UNIX и Linux семейство компрессоров представлено сразу несколькими программами: **compress**, **gzip**, **bzip2**. Они используют разные алгоритмы сжатия и в разной степени эффективны. Каждый компрессор включает три утилиты: утилита сжатия (**compress**, **gzip**, **bzip2**), утилита декомпрессии (**uncompress**, **gunzip**, **bunzip2**) и еще одна утилита декомпрессии, которая по умолчанию выдает результат в стандартный вывод. Утилиты разных компрессоров имеют унифицированный интерфейс, так что нет необходимости запоминать ключи для каждого компрессора отдельно.

gzip

gzip — это GNU-версия программы **compress**, созданная, чтобы избежать патентных ограничений. Ключи у **gzip** такие же, как у **compress**. Дополнительно есть полезная опция, задающая с помощью числового значения от 1 до 9 степень сжатия. Это позволяет управлять соотношением сжатие / время в зависимости от потребностей. Например, можно задать -5 для больших файлов и выиграть на времени исполнения при достаточной степени сжатия. Суффиксы сжатых файлов, которые добавляет **gzip**, — **.gz**. Для декомпрессии есть утилиты **gunzip** и **gzcat**, аналогичные **uncompress** и **zcat**. **gunzip**. Ищут файлы с суффиксами **.gz** и **.Z**, **.z**, **tgz**, **taz**.

```
gzip [ключи] [файл]...
```

-c — направляет результат в стандартный вывод, исходные файлы не меняются. Обычно используется совместно с архиватором.

-d — выполнить декомпрессию указанных файлов.

Без ключей **gzip** выполняет сжатие каждого файла из заданного списка и записывает результат в новый файл, добавляя к имени суффикс **.gz**. Исходные несжатые файлы удаляются.

Пример использования совместно с архиватором:

```
tar cf - ./data | gzip -c > data.tar.gz
```

gunzip

Декомпрессия с **gunzip**:

```
gunzip [файл]...
```

Выполняет декомпрессию файлов из списка, если надо, добавляя к именам суффикс **gz**. Разархивированный файл записывается с именем без **.gz**. Исходный сжатый файл удаляется.

zcat

Декомпрессия с выдачей в стандартный вывод **zcat**:

```
zcat [файл]...
```

Для декомпрессии сжатых zip-архивов:

```
tar xzf file.tar.gz
```

Или еще **zcat** используют совместно с архиватором для декомпрессии сжатых архивов:

```
zcat file.tar.gz | tar xf -
```

Еще один популярный компрессор — `bzip2`. Превосходит на большинстве файлов `gzip` по эффективности сжатия, но проигрывает в скорости. Опции такие же, как у `gzip`. Расширения архивов — `.bz2`.

Практическое задание

1. Просмотреть содержимое директорий `/etc`, `/proc`, `/home`. Посмотреть пару произвольных файлов в `/etc`.
2. Выяснить, для чего предназначена команда `cat`. Используя данную команду, создать два файла с данными, а затем объединить их. Просмотреть содержимое созданного файла. Переименовать файл, дав ему новое имя.
3. Создать несколько файлов. Создать директорию, переместить файл туда. Удалить все созданные в этом и предыдущем задании директории и файлы.
4. В ОС Linux скрытыми файлами считаются те, имена которых начинаются с символа `."`. Сколько скрытых файлов в вашем домашнем каталоге? (Использовать конвейер. Подсказка: для подсчета количества строк можно использовать `wc`.)
5. Попробовать вывести с помощью `cat` содержимое всех файлов в директории `/etc`. Направить ошибки в отдельный файл в вашей домашней директории. Сколько файлов, которые не удалось посмотреть, оказалось в списке?
6. Запустить в одном терминале программу, а в другом терминале посмотреть PID процесса и остановить с помощью `kill`, посылая разные типы сигналов. Что происходит?
7. * Отобразить в `/dev` список устройств, которые в настоящее время используются вашим UID (для этого используется команда `lsdf`). Организовать конвейер через `less`, чтобы посмотреть их должным образом.
8. * Создать директорию для хранения фотографий. В ней должны быть директории по годам (например, за последние 5 лет), и в каждой директории года — по директории для каждого месяца.
9. * Заполнить директории файлами вида `ГГГГММДДНН.txt`. Внутри файла должно быть его имя. Например: `2018011301.txt`, `2018011302.txt`.
10. * Полезное задание на конвейер. Использовать команду `cut` на вывод длинного списка каталога, чтобы отобразить только права доступа к файлам. Затем отправить в конвейере этот вывод на `sort` и `uniq`, чтобы отфильтровать все повторяющиеся строки. Потом с помощью `wc` подсчитать различные типы разрешений в этом каталоге. Самостоятельно решить задачу, как сделать так, чтобы в подсчет не добавлялись строка «Итого» и файлы `.` и `..` (ссылки на текущую и родительскую директорию).

Примечание. Задания с 7 по 10 даны для тех, кому упражнений 1–6 показалось недостаточно.

Дополнительные материалы

1. [MobaXterm](#).
2. [Настройка Ubuntu 18.04 после установки](#).

3. [Хитрости работы в Ubuntu 16–18.](#)
4. [Раскладка клавиатуры в Ubuntu 18.](#)
5. [Поиск пакетов в Ubuntu.](#)
6. [Персональные архивы пакетов \(PPA\).](#)
7. [Установка Unity.](#)
8. [В Windows тоже есть перенаправления, пайпы, свой /dev, /dev/null, /etc/fstab/.](#)

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. [Философия Unix.](#)
2. [Раскладка клавиатуры в Ubuntu.](#)
3. [Руководство пользователя Ubuntu.](#)