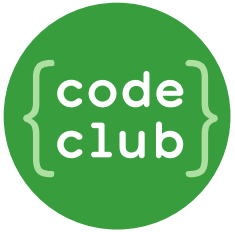
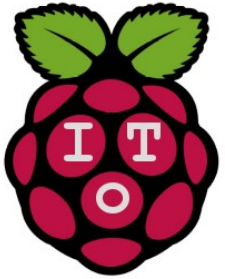


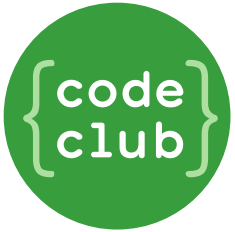
Internet of Things



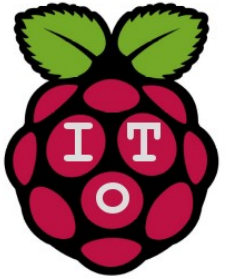
Автоматизация с помощью командных файлов

Шадринск
2018-2019

М. В. Шохирев



Командный язык bash



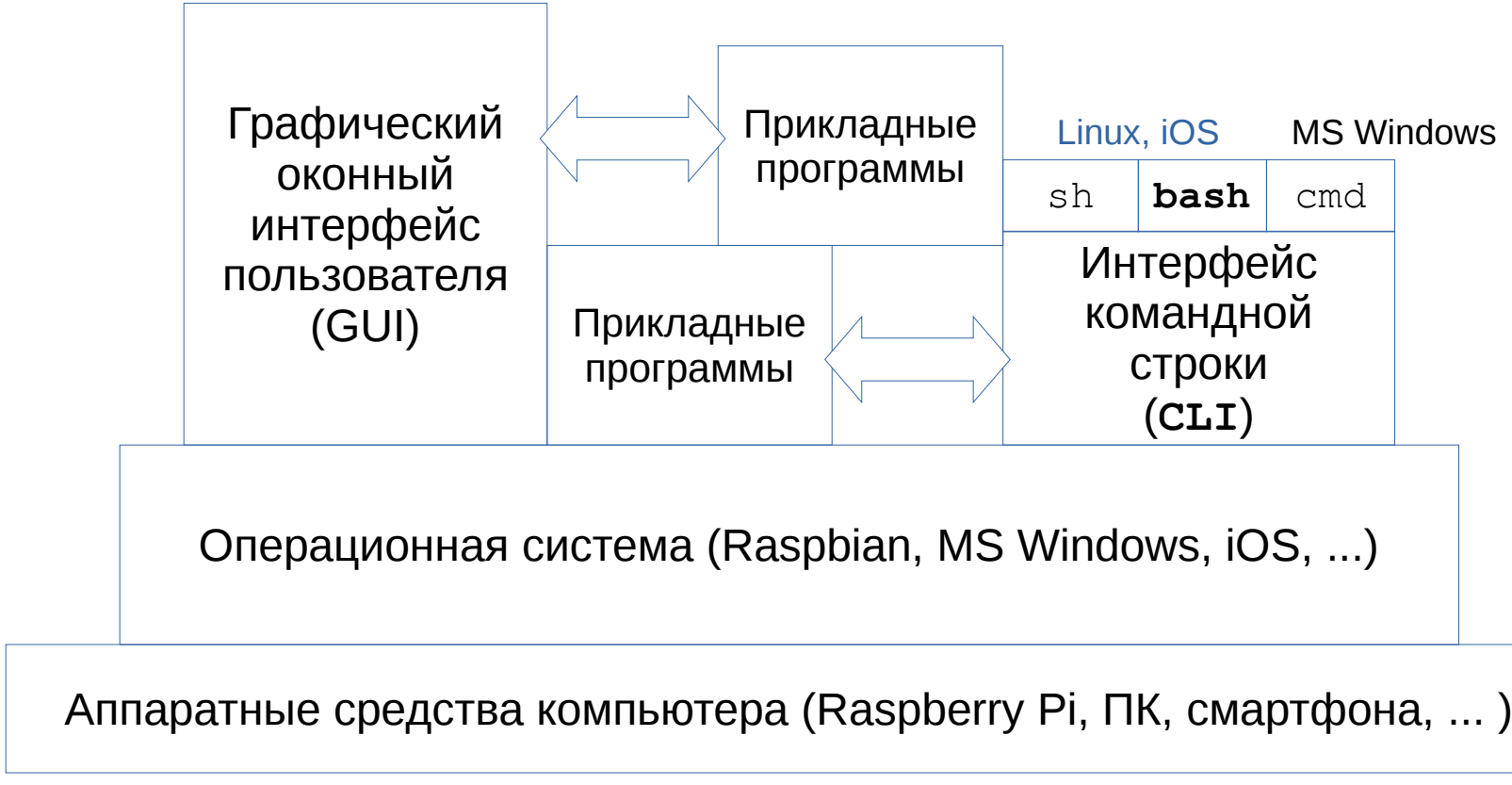
В GNU/Linux и других Unix-подобных ОС многие задачи по автоматизации выполнения самых разных действий выполняются при помощи **командных файлов**, написанных на языках сценариев (*скриптовых языках*): **`Perl`**, **`Python`**, **`Ruby`**, **`AWK`**, **`Tcl`** и других, а также на языке командной оболочки (*shell*) ОС.

Чаще других применяется современный shell-интерпретатор **`bash`**, который имеет богатые средства программирования.

Командные файлы (*скрипты*) основаны на вызовах *команд* ОС Unix, выполняющих нужные действия.

Взаимодействие с ОС через команды называется **`CLI`** = Command Line Interface.

Командная оболочка



Зачем командный язык?

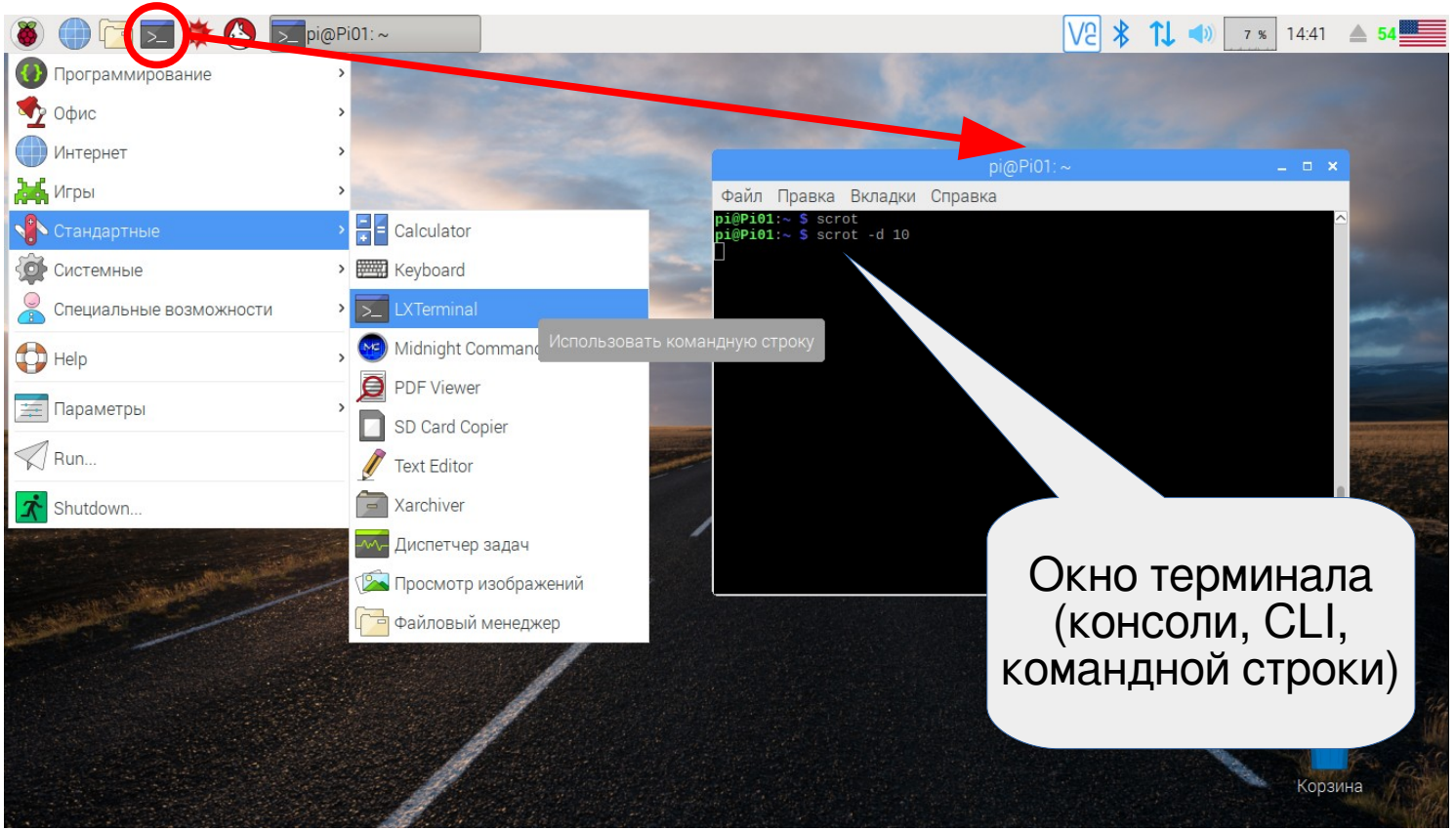
Почему нельзя всё делать с помощью графического интерфейса (GUI)?

Можно, но это очень неэффективно! Потому, что:

- Действия в GUI делает человек, а ему это надоедает. Он медлителен. И ещё он обязательно иногда ошибается!
- Действия должны выполняться быстро.
- Действия должны выполняться безошибочно.
- Действия должны повторяться многократно.
- Действия должны выполняться постоянно: ночью, в выходные, ...
- Действия должны выполняться автоматически!
- То есть без вмешательства человека.

Компьютер должен работать, а человек — думать!

CLI = интерфейс командной строки



В **Raspbian** (и других ОС на основе **Linux**) параллельно с графической оболочкой включен *терминальный доступ* (интерфейс командной строки = **CLI**) с текстовых консолей, на которые можно переключаться по сочетанию клавиш **Ctrl+Alt+F1..F6** Чтобы вернуться в графическую среду, нажмите клавиши **Ctrl+Alt+F7**.

Всё — это файл

В ОС Unix / Linux любая информация обычно представлена в виде *файла*:

- Обычные файлы: текстовые, двоичные (фото, аудио, видео, архивы, ...), ...
- Каталоги (папки) — это специальные файлы с данными о файлах (имя, ...)
- Символические ссылки (на файлы и каталоги)
- Файлы конфигурации в **`/etc/`**
- Устройства в **`/dev/`** – с посимвольным и блочным доступом
- Внешние носители данных: диски, карты памяти, ...
- Сетевые ресурсы с других компьютеров в сети
- Сетевые соединения – **`sockets`**
- Выполняющиеся в оперативной памяти процессы в **`/proc/`**
- Системная информация из ядра (kernel) ОС в **`/sys/`**
- Виртуальные файловые системы (VFS): NFS, sshFS, ISO, davfs2 (Yandex disk), ...
- ...

В виде файлов хранятся многие данные о «железе», например, показания встроенного датчика температуры, который постоянно измеряет нагрев системной микросхемы (SoC = System-On-a-Chip).

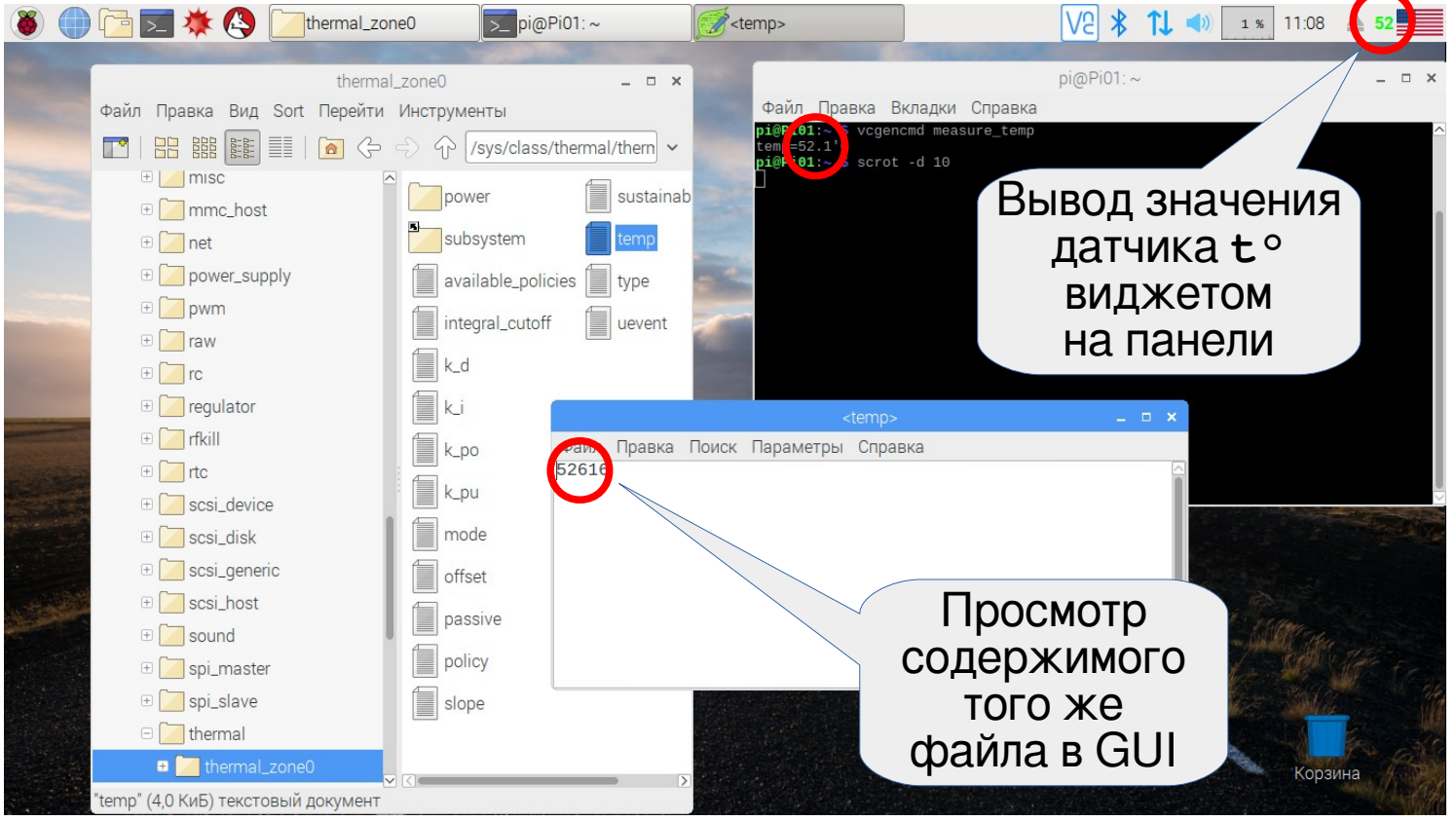
Температура SoC

```
cat /sys/class/thermal/thermal_zone0/temp
```

Просмотр содержимого файла командой

Команда вывода значения встроенного датчика t°

```
vcgencmd measure_temp
```



Полезные команды

Команды оболочки

Unix-подобных операционных систем

Формат команд:

команда -ключ1 -ключ2 -ключN параметр1 параметр2 параметрN

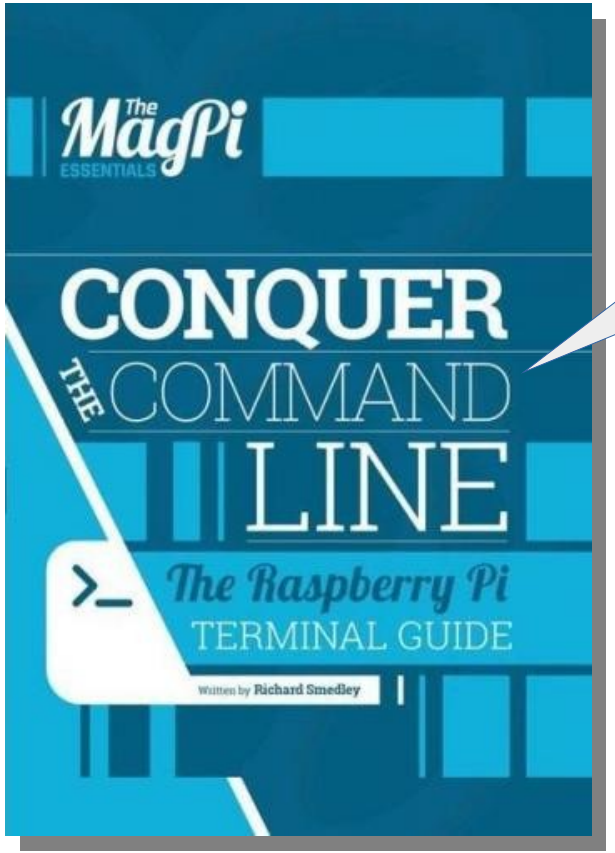
Элементы команды разделяются пробелами; если элемент включает пробелы, его заключают в 'одинарные кавычки', а если требуется подстановка по шаблону, то в "двойные кавычки".

Специальные обозначения в командах:

Описание	Обозначение	Пример	Пояснения
Разделитель элементов команды	пробел	cp 1.txt /tmp	Скопировать 1.txt в /tmp
«Ключ» / «флаг» = указание / режим работы команды	*-флаг* или *--режим*	ls -a	-a = «All» - все файлы
Корень файловой системы	/	cd /	Рабочим каталогом станет /
Скрытый файл или каталог	*.имя*	cd .config	Видны по ls -a
Текущий каталог	.	ls .	То же, что просто ls
Родительский каталог (каталог на 1 уровень выше)	..	cd ..	Рабочим станет
Домашний каталог (папка) пользователя	~	cd ~	
Шаблон для одного любого символа в имени	?	cat ?.txt	
Шаблон для нескольких любых символов в имени	*	cat *.txt	1.txt a.tx
Отмена специального значения символа после \	\	mkdir Мои\ файлы	То же, что m
Перенаправить вывод команды в файл (записать в файл)	*команда > файл*	cat ?.txt > all.txt	all.txt=1.txt+
Перенаправить вывод команды в файл (дописать в файл)	*команда >> файл*	cat ??.txt >> all.txt	all.txt=all.txt
Направить данные из файла на вход команды	*команда < файл*	sort < 1.txt	То же, что ca

Справка
по основным командам
Unix / Linux
в отдельном файле:
IoT-Shell_commands.pdf

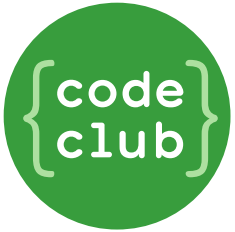
Учебник по командам



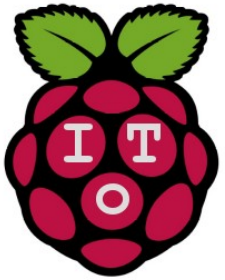
Учебник
по работе
в командной строке
на **Raspberry Pi**

Для углублённого изучения, как работать в командной строке, подойдут любые многочисленные ресурсы в сети Internet на русском языке, посвящённые **sh**, **bash** или **CLI** в ОС Linux.

https://www.raspberrypi.org/magpi-issues/Essentials_Bash_v1.pdf



Информация о системе



Температура системной микросхемы (SoC = System-On-a-Chip):

```
vcgencmd measure_temp
cat /sys/class/thermal/thermal_zone0/temp
```

Сведения о центральном процессоре (CPU):

```
cat /proc/cpuinfo
```

Сведения о частоте CPU (минимальной, максимальной, текущей):

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

Сведения о свободной оперативной памяти (RAM):

```
free -h
```

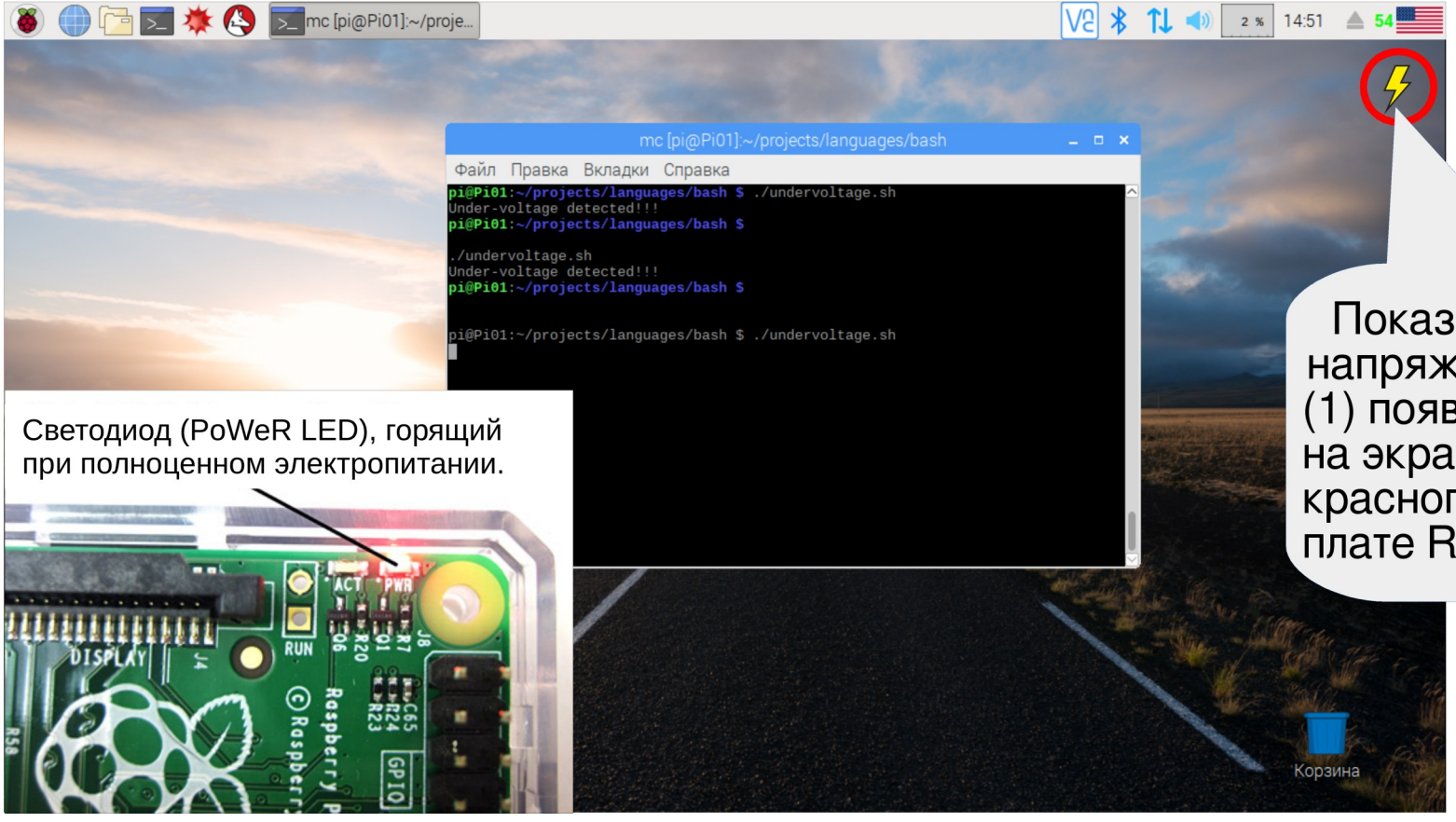
Сведения о распределении памяти между CPU и GPU (графическим процессром):

```
vcgencmd get_mem arm
vcgencmd get_mem gpu
```

Сведения о свободном пространстве на разделах карты памяти, USB-дисков:

```
df -h
```

Ситуация: низкое напряжение

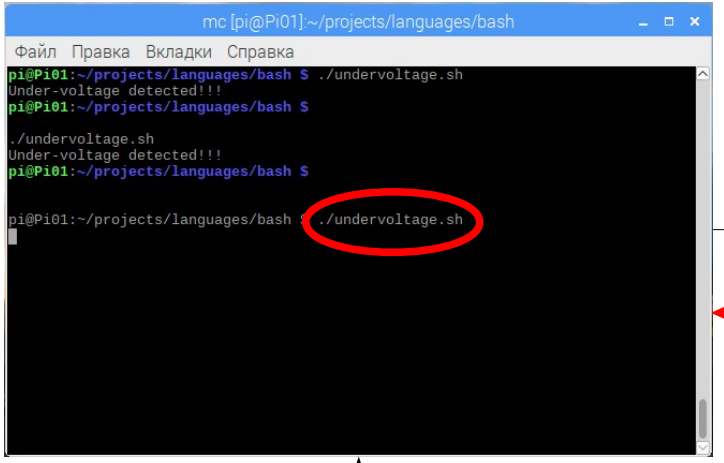


Низкое напряжение: программный контроль

Скрипт на **bash** для проверки низкого напряжения (< 5V) от блока питания и реагирования на ситуацию.
Из системного файла **/sys/class/leds/led1/brightness** считывается значение яркости светодиода электропитания (*255 = включен; 0 = выключен*). Его периодическое выключение означает критическое понижение напряжения (< 4.5V):

```
#!/bin/bash
while true # повторение в цикле, пока true, т. е. бесконечно
do
  POWER_LED=`cat /sys/class/leds/led1/brightness` # чтение значения
  if [ $POWER_LED = 0 ]; then # проверка значения в переменной на 0
    echo Обнаружено низкое напряжение!!! # реакция на U < 5V
    sleep 1 # задержка выполнения на 1 секунду
  fi
done
```

bash: как это работает?



В терминальном окне из командной строки запускается на выполнение скрипт, в 1-й строке которого вызывается интерпретатор **bash.**

Считывает строки из командного файла и выполняет каждую из них

Интерпретатор **bash**

Сообщения выводятся в окно терминала:

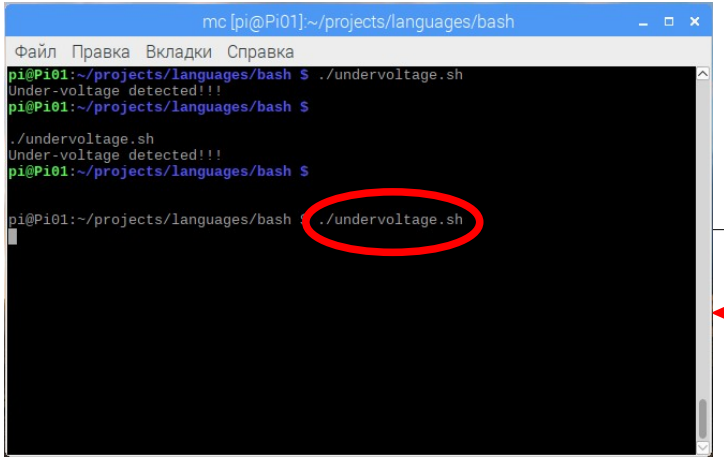
echo Низкое напряжение!

```
#!/bin/bash
while true
do
  POWER_LED=`cat /sys/class/leds/led1/brightness`
  if [ $POWER_LED = 0 ]; then
    echo Обнаружено низкое напряжение!!!
    sleep 1
  fi
done
```

Командный файл (скрипт)

Скрипт (программа) состоит из вызовов команд (в том числе других скриптов) и программных конструкций, чтобы контролировать ход их выполнения (условия, циклы и т. п.)

Перенаправление ввода-вывода



В терминальном окне из командной строки запускается на выполнение скрипт, в 1-й строке которого вызывается интерпретатор **bash.**

Считывает строки из командного файла и выполняет каждую из них

Интерпретатор **bash**

Сообщения выводятся в окно терминала:

```
echo Низкое напряжение!
```

Стандартный вывод (**STDOUT**) любой программы по умолчанию связан с терминалом (вывод на экран), но его можно *перенаправить*, чтобы он записывался в файл с начала (**`> file`**) или дозаписывался в конец файла (**`>> file`**).

Перенаправление ввода-вывода

Стандартный вывод (1, **STDOUT**) любой программы изначально связан с терминалом (вывод на экран), но его можно *перенаправить*, чтобы он записывался в файл с начала (**> file**) или дозаписывался в конец файла (**>> file**).

```
script 1>  out_file
script  >  out_file
script 1>> out_file
script  >> out_file
```

Стандартный протокол (2, **STDERR**) любой программы по умолчанию связан с терминалом (вывод на экран), но его можно *перенаправить*, чтобы он записывался в файл с начала (**> file**) или дозаписывался в конец файла (**>> file**).

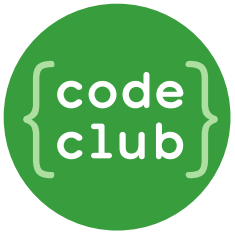
```
script 2>  err_file
script 2>> err_file

script > file 2>&1
script &> file
```

Стандартный ввод (0, **STDIN**) любой программы по умолчанию связан с терминалом (ввод с клавиатуры), но его можно *перенаправить*, чтобы он считывался из файла (**< file**).

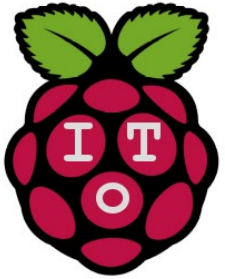
```
script < in_file
```

Стандартный вывод (1, **STDOUT**) любой программы можно *перенаправить* на стандартный ввод (0, **STDIN**) другой программы (**script1 | script2**).

```
script1 | script2
```



Конвейер команд



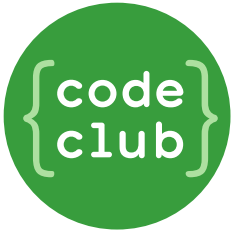
Сочетая разные способы *перенаправления ввода-вывода* можно записать *конвейер команд*, в котром каждая команда будет принимать данные, поступившие из *стандартного ввода* (от предыдущей команды), обрабатывать их и через свой *стандартный вывод* передавать на обработку следующей команде. А последняя команда запишет обработанные данные в выходной файл.

```
script1 < in_file | script2 | script3 > out_file
```

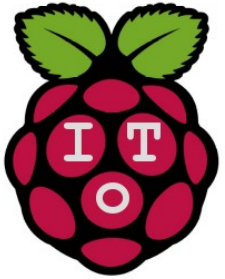
На этом примере демонстрируются крайне полезные приёмы работы в ОС **Unix**:

- разрабатывать программы, каждая из которых хорошо выполняют одну функцию
- и хорошо сочетается с другими, тогда
- последовательная обработка несколькими такими программами
- может решить сложную задачу.

```
kill `ps ax | grep puma | grep 3000 | cut -b 1-6`
```



Программный контроль: запуск по расписанию

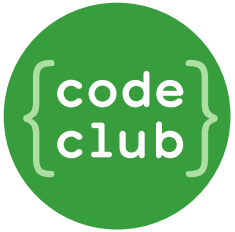


Чтобы скрипты *программного контроля* различных ситуаций выполнялись регулярно по расписанию, нужно заполнить настроечный файл (таблицу-расписание) для «демона» (службы) **cron**, который занимается периодическим выполнением заданий в назначенное время. Это делается командой **crontab**:

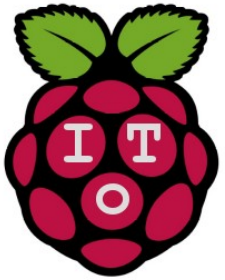
crontab -e	изменить (Edit) таблицу (пользователя pi)
crontab -l	просмотреть (List) таблицу (пользователя pi)
sudo crontab -l -u root	просмотреть таблицу пользователя root

Запуск по расписанию: crontab

```
# формат строк в таблице расписания crontab:
# m h DoM Mon DoW command
# * * *   *   *   выполняемая команда
# ┬ ┬ ┬   ┬   ┬
# │ │ │   │   └──── Day of Week/день недели: 1..7 = Пн..Вс
# │ │ │   └─────── Month/месяц: 1..12 = январь..декабрь
# │ │ └──────────── Day of Month/день месяца: 1..31
# │ └────────────── Hour/часы: 0..23
# └──────────────── Minute/минуты: 0..59

*/5  *  *  *  *     /home/pi/bin/every5    # каждые 5 минут ежедневно
  0  0  *  *  6,7   /home/pi/bin/weekend   # в полночь по выходным
 0 */2  *  *  1-5   /home/pi/bin/workdays  # каждые 2 часа по рабочим дням
 23 59  1  *  *     /home/pi/bin/monthly   # в 23:59 1 числа месяца
@reboot             /home/pi/bin/at_boot   # при каждой загрузке
```



Запуск по расписанию: crontab

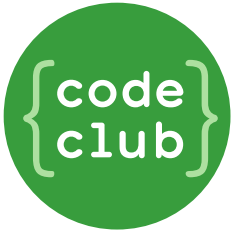


Вместо первых 5 полей с указанием времени выполнения команды

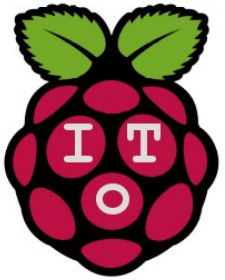
```
# m h DoM Mon DoW command
  * * *   *   *
```

можно применять специальные именованные указания:

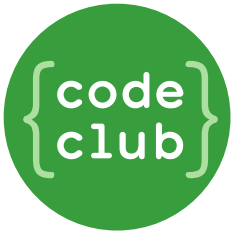
```
@reboot      /home/pi/bin/at_boot       # при каждой загрузке

@hourly      /home/pi/bin/every_hour    # ежечасно = "0 * * * *"

@daily       /home/pi/bin/every_day     # ежедневно = "0 0 * * *"
@midnight    /home/pi/bin/at_midnight   # в полночь = @daily

@weekly      /home/pi/bin/every_week    # по воскресеньям = "0 0 * * 0"
@monthly     /home/pi/bin/every_month   # 1-го числа месяца = "0 0 1 * *"

@yearly      /home/pi/bin/every_year    # 1 января = "0 0 1 1 *"
@annually    /home/pi/bin/every_year    # ежегодно = @yearly
```



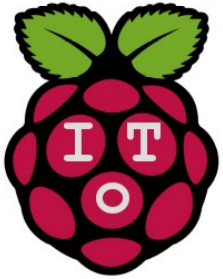
Программный контроль: запуск по расписанию



Как вы думаете,
какие действия
удобно автоматически выполнять
по расписанию?



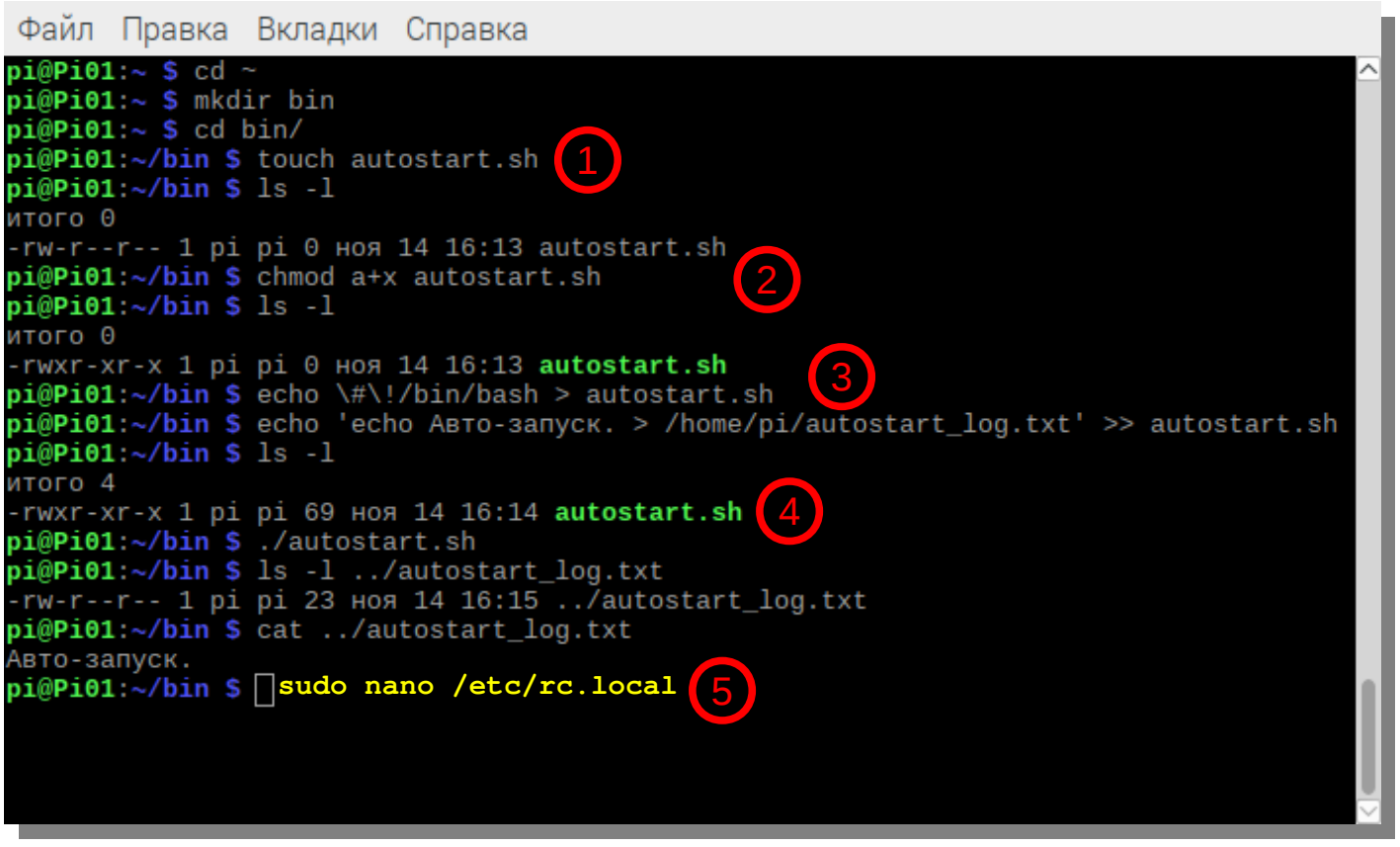
Программный контроль: автозапуск



Чтобы скрипты *программного контроля* различных ситуаций выполнялись постоянно, нужно автоматически запускать их после включения Raspberry Pi и загрузки ОС.

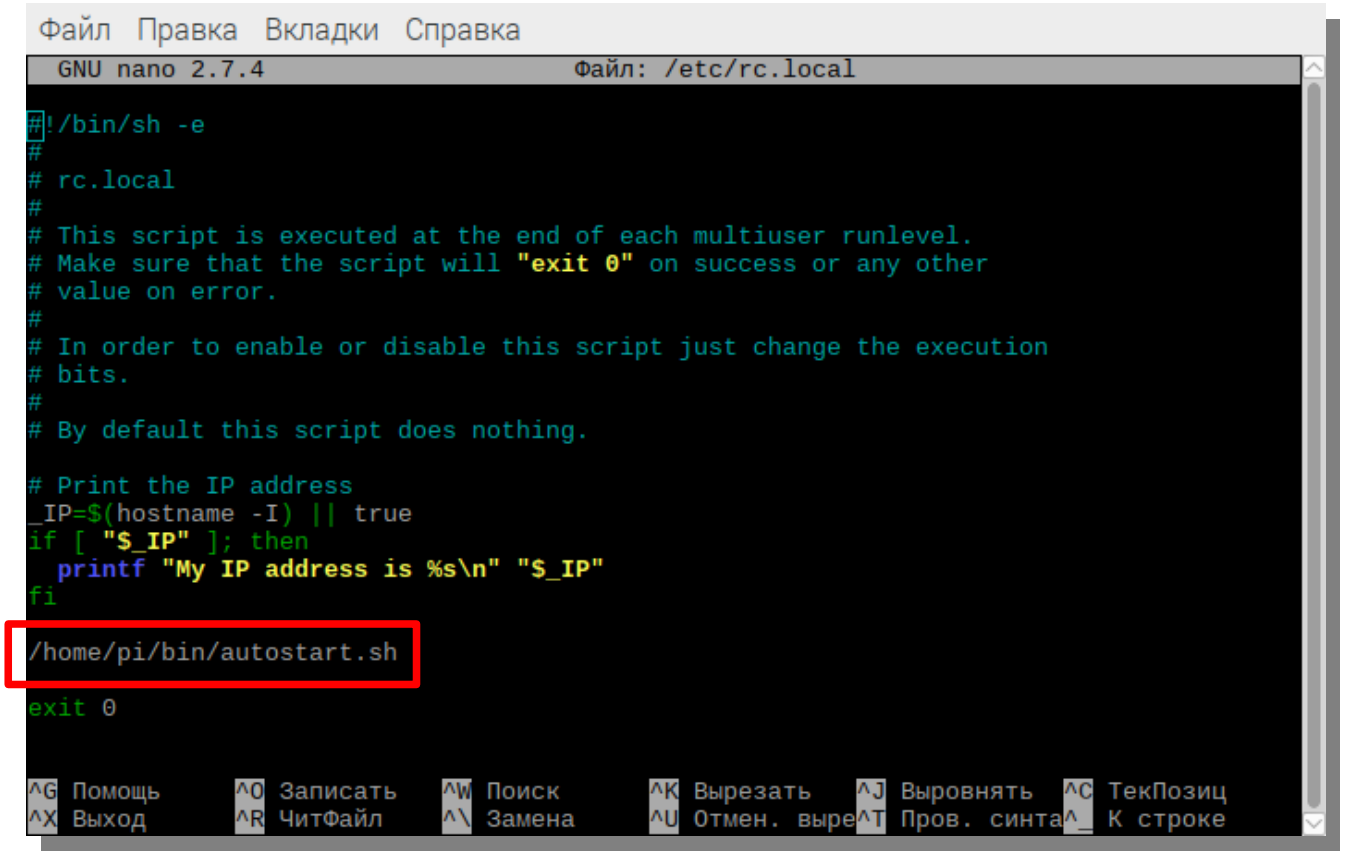
`/etc/rc.local`	*системный* скрипт авто-запуска во время загрузки ОС, куда можно добавить свои команды, например, пользовательский скрипт.
`/home/pi/bin/autostart.sh`	пользовательский скрипт, который требуется запускать автоматически, куда можно добавлять свои команды.

Автозапуск: настройка

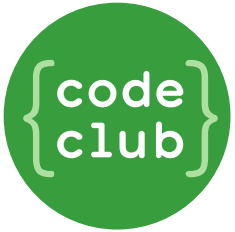


1. Создать скрипт "**autostart.sh**", который будет запускаться при загрузке.
2. Назначить ему признак «исполняемый».
3. Добавить в него нужные команды для выполнения.
4. Попробовать выполнить его и проверить результат работы.
5. Запустить от имени суперпользователя (**sudo**) текстовый редактор **nano**, чтобы изменить системный конфигурационный файл **/etc/rc.local**.

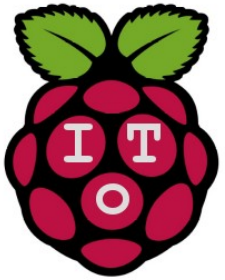
Автозапуск: /etc/rc.local



6. Добавить в системном конфигурационном файле **`/etc/rc.local`** строку с именем командного файла для выполнения.



Автозапуск: действия при включении



Как вы думаете,
какие действия
удобно автоматически выполнять
при (пере)запуске компьютера?