

Взаимодействие bash-скриптов с пользователем

- *nix
Любой приказ, который может быть неправильно понят, понимается неправильно (Армейская аксиома)

Редкий скрипт лишен необходимости общения с пользователем. Мы ожидаем, что программа (утилита) будет выполнять то, что нам от нее хочется. Следовательно, нужны инструменты влияния на них, да и программа сама должна объяснить, как продвигается ее работа.

Данным топиком я хочу рассмотреть несколько способов взаимодействия bash-скриптов с пользователем. Статья рассчитана на новичков в скриптинге, но, надеюсь, люди опытные тоже найдут что-нибудь интересное для себя.

Топик так же снабжен примитивными примерами, не несущими смысловой нагрузки, но позволяющими посмотреть в работе некоторые интересные штуки.

Переменные

Самый распространенный способ хранения начальных данных — переменные. В самом начале программы объявляются несколько таких переменных, в которые пользователь записывает некоторые исходные данные.

```
#!/bin/bash
```

```
# Вписать сюда адрес электронной почты
```

```
EMAIL=example@gmail.com
```

```
echo "Адрес электронной почты: $EMAIL"
```

Такой способ хорош, если данных не много и скрипт рассчитан на автоматическое выполнение без участия пользователя. Необходимо ясно известить пользователя о том, что и где ему необходимо вписать. Желательно собрать все это в одном месте — [файле конфигурации](#). Подключить его можно командой **source**.

Например, если конфигурационный файл будет лежать в той же директории, что и скрипт, мы получим:

```
#!/bin/bash
```

```
source ./config.cfg
```

```
echo "Адрес электронной почты: $EMAIL"
```

В файл **config.cfg** не забудем поместить строчку `EMAIL=example@gmail.com`

Параметры командной строки

Еще один способ сообщить данные программе — указать при запуске в командной строке. Содержатся эти параметры в переменных с номерами. Например: **\$0** — имя скрипта, **\$1** — первый параметр, **\$2** — второй параметр и т. д. Также существуют две вспомогательные переменные: **\$#** содержит количество переданных аргументов; **\$@** содержит все аргументы, переданные скрипту, разделенные пробелами.

```
#!/bin/bash
```

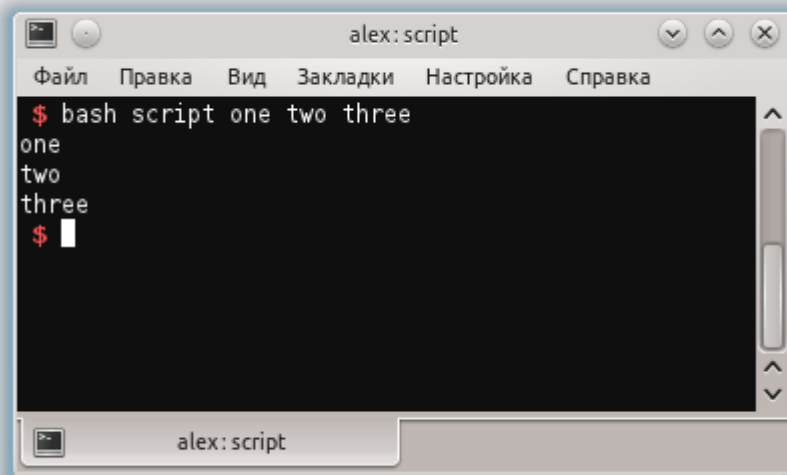
```
# Цикл выдаст все переданные аргументы
```

```
for n in $@
```

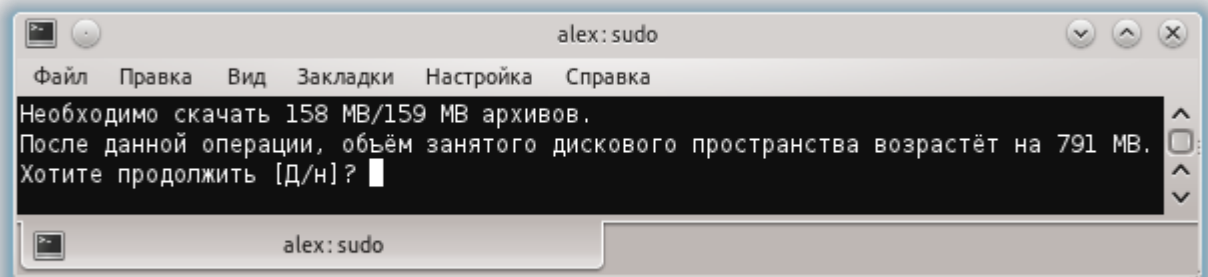
```
do
```

```
    echo "$n"
```

```
done
```



Вопросы и подтверждения



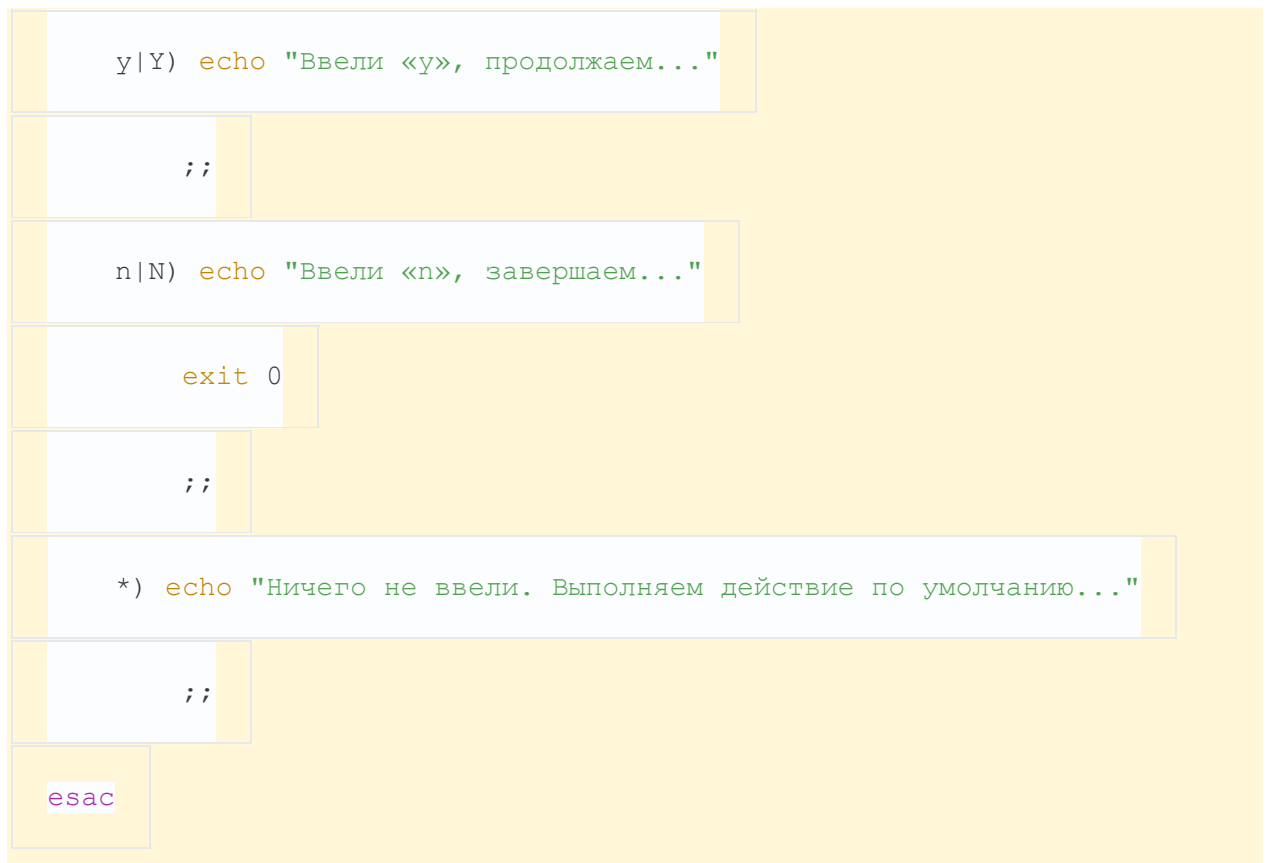
Думаю многим знаком вопрос со скриншота выше. Такой диалог можно использовать... ну вы и сами догадались, где его можно использовать.

```
#!/bin/bash
```

```
echo -n "Продолжить? (y/n) "
```

```
read item
```

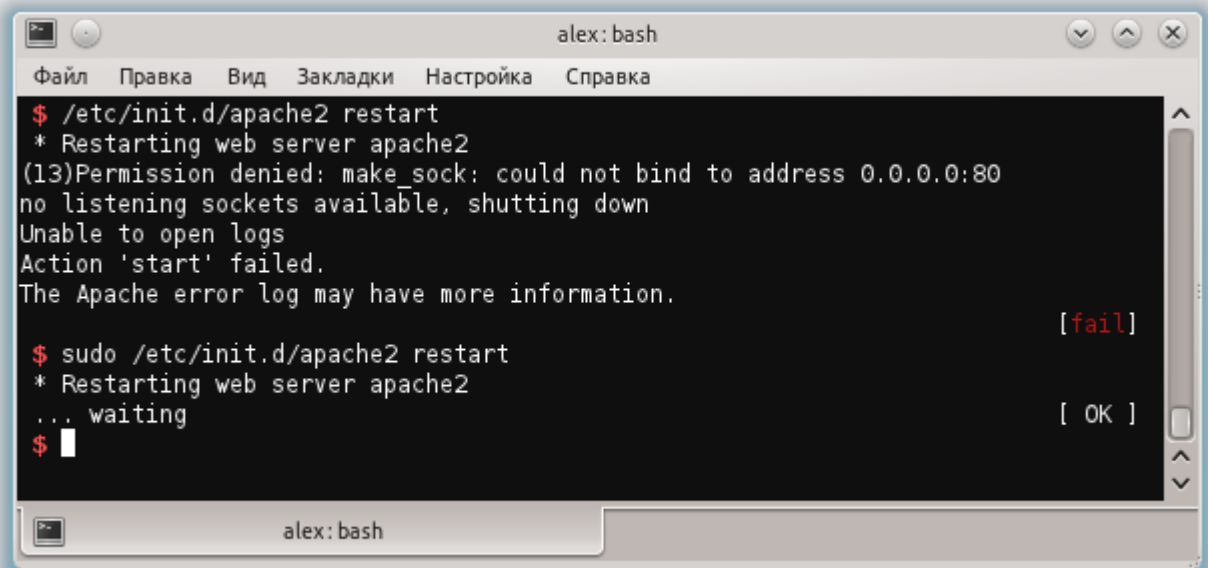
```
case "$item" in
```



Обратите внимание, что на скриншоте буква «Д» — большая. Это означает действие по умолчанию, то есть если пользователь ничего не введет, то это будет равнозначно вводу «Д».

OK / FAIL

Еще одним способом общения программы с пользователем являются статусы выполнения. Скорее всего они вам знакомы.



```
alex: bash
Файл  Правка  Вид  Закладки  Настройка  Справка
$ /etc/init.d/apache2 restart
* Restarting web server apache2
(13)Permission denied: make_sock: could not bind to address 0.0.0.0:80
no listening sockets available, shutting down
Unable to open logs
Action 'start' failed.
The Apache error log may have more information.
[fail]
$ sudo /etc/init.d/apache2 restart
* Restarting web server apache2
... waiting
[ OK ]
$
```

Реализация тоже довольно проста.

```
#!/bin/bash
```

```
SETCOLOR_SUCCESS="echo -en \033[1;32m"
```

```
SETCOLOR_FAILURE="echo -en \033[1;31m"
```

```
SETCOLOR_NORMAL="echo -en \033[0;39m"
```

```
echo -e "Удаляется файл..."
```

```
# Команда, которую нужно отследить
```

```
rm test_file
```

```
if [ $? -eq 0 ]; then
```

```
    $SETCOLOR_SUCCESS
```

```

echo -n "$(tput hpa $(tput cols))$(tput cub 6) [OK]"

$SETCOLOR_NORMAL

echo

else

$SETCOLOR_FAILURE

echo -n "$(tput hpa $(tput cols))$(tput cub 6) [fail]"

$SETCOLOR_NORMAL

echo

fi

```

Вот так выглядит работа скрипта:

```

alex: bash
Файл  Правка  Вид  Закладки  Настройка  Справка
$ bash script
Удаляется файл...
rm: невозможно удалить «test_file»: Нет такого файла или каталога
[fail]
$ touch test_file
$ bash script
Удаляется файл...
[OK]
$

```

Хорошие люди написали расширенную версию скрипта с логированием и прогрессом выполнения. С радостью поделюсь [ссылкой](#).

Исходя из вышеприведенной ссылки код можно упростить.

```
#!/bin/bash
```

```
red=$(tput setf 4)
```

```
green=$(tput setf 2)
```

```
reset=$(tput sgr0)
```

```
toend=$(tput hpa $(tput cols))$(tput cub 6)
```

```
echo -e "Удаляется файл..."
```

```
# Команда, которую нужно отследить
```

```
rm test_file
```

```
if [ $? -eq 0 ]; then
```

```
    echo -n "${green}${toend}[OK]"
```

```
else
```

```
    echo -n "${red}${toend}[fail]"
```

```
fi
```

```
echo -n "${reset}"
```

```
echo
```

Псевдографика

Для любителей графического представления существуют удобный инструмент: **dialog**. По умолчанию его в системе нет, так что исправим положение.

```
sudo apt-get install dialog
```

Опробовать его можно простой командой:

```
dialog --title " Уведомление " --msgbox "\n Свершилось что-то
```

```
страшное!" 6 50
```

Вот пример диалога прогресса:

```
#!/bin/sh
```

```
(
```

```
c=10
```

```
while [ $c -ne 110 ]
```

```
do
```

```
echo $c
```

```
((c+=10))
```

```
sleep 1
```

```
done
```

```
) |
```



```
dialog --title " Тест диалога прогресса " --gauge "Please wait ...."
```

```
10 60 0
```

```
clear
```

Не забываем вставлять **clear** для очистки экрана, чтобы не оставлять синий фон. Эта утилита поддерживает еще очень много типов диалоговых окон. Главным недостатком является то, что по умолчанию ее нет в системе.

Альтернативой **dialog** может служить **whiptail**, который даже присутствует в некоторых системах по умолчанию.

Подробнее можно ознакомиться по ссылкам:

<http://unstableme.blogspot.com/2009/12/linux-dialog-utility-short-tutorial.html>

<http://www.cc-c.de/german/linux/linux-dialog.php>

GUI

Хоть есть ярые противники [GUI](#), но он явно имеет право на существование. Такие диалоги можно получить с помощью команды **kdialog** (если графической оболочкой выступает KDE), либо **gdialog** и **zenity** (для Gnome).

Например, форма для ввода пароля:

```
kdialog --password "Пожалуйста, введите свой пароль:"
```

либо

```
gdialog --password "Пожалуйста, введите свой пароль:"
```

Еще пример один для KDE:

```
kdialog --question "Вы хотите продолжить?"
```

```
rc=$?
```

```
if [ "${rc}" == "0" ]; then
```

```
echo "Нажали yes"

else

echo "Нажали no"

fi
```

И для Gnome:

```
#!/bin/bash

name=$(gdialog --title "Ввод данных" --inputbox "Введите ваше имя:")

50 60 2>&1)

echo "Ваше имя: $name"
```

Как видите, явным недостатком этого метода является привязанность к конкретной среде рабочего стола. Да и вообще к графической среде, которая может и отсутствовать. Но, тем не менее, может и пригодиться когда-нибудь.

Подробнее по ссылкам:

<http://pwet.fr/man/linux/commandes/kdialog>

http://linux.about.com/library/cmd/blcmdl1_gdialog.htm

<http://www.techrepublic.com/blog/opensource/gui-scripting-in-bash/1667>

P.S. Продолжение следует...

UPD: Добавил упрощенный код в раздел «ОК / FAIL».

UPD2: Добавил пример подключения конфигурационного файла в раздел «Переменные».

Опубликована [вторая часть](#).

Взаимодействие bash-скриптов с пользователем. Часть 2

- [*nix](#)

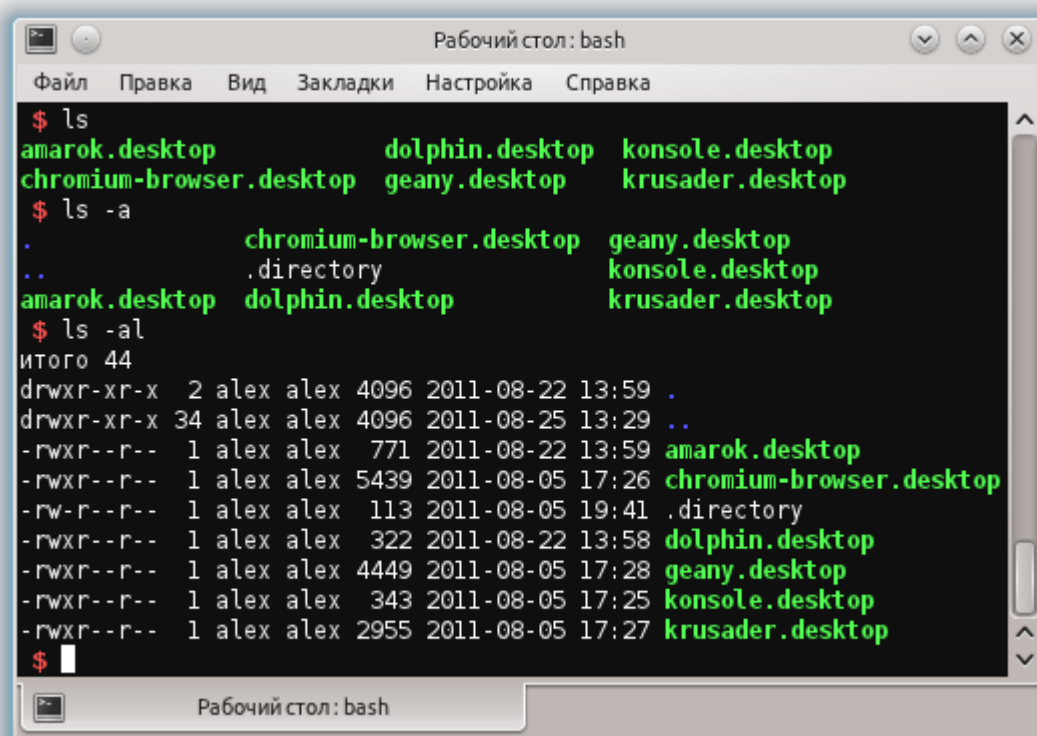
Наша программа настолько сурова, что даже логин отображается звездочками (bash.org.ru)

Вашему вниманию представляется новая подборка средств общения скриптов с пользователем. Надеюсь, интересно будет всем, кто не боится работать с консолью.

Первую часть можно найти [тут](#).

Опции (ключи)

Этот способ был достоин первой части статьи, но не попал туда из-за забывчивости автора. Несомненно, этот способ знаком всем пользователям *nix, хоть раз работавших с консолью. Простой наглядный пример:



```
Рабочий стол: bash
Файл  Правка  Вид  Закладки  Настройка  Справка
$ ls
amarok.desktop      dolphin.desktop    konsole.desktop
chromium-browser.desktop  geany.desktop      krusader.desktop
$ ls -a
.                   chromium-browser.desktop  geany.desktop
..                  .directory               konsole.desktop
amarok.desktop     dolphin.desktop         krusader.desktop
$ ls -al
итого 44
drwxr-xr-x  2 alex alex 4096 2011-08-22 13:59 .
drwxr-xr-x 34 alex alex 4096 2011-08-25 13:29 ..
-rwxr--r--  1 alex alex  771 2011-08-22 13:59 amarok.desktop
-rwxr--r--  1 alex alex 5439 2011-08-05 17:26 chromium-browser.desktop
-rw-r--r--  1 alex alex  113 2011-08-05 19:41 .directory
-rwxr--r--  1 alex alex  322 2011-08-22 13:58 dolphin.desktop
-rwxr--r--  1 alex alex 4449 2011-08-05 17:28 geany.desktop
-rwxr--r--  1 alex alex  343 2011-08-05 17:25 konsole.desktop
-rwxr--r--  1 alex alex 2955 2011-08-05 17:27 krusader.desktop
$
```

Плюс в том, что ключи короткие и их можно комбинировать. Попробуем сделать что-нибудь подобное, а заодно изучим еще несколько моментов.

```
#!/bin/bash
```

```
set -e
```

```
ME=`basename $0`
```

```
function print_help() {
```

```
echo "Работа с файлом test_file"
```

```
echo
```

```
echo "Использование: $ME options..."
```

```
echo "Параметры:"
```

```
echo "  -c          Создание файла test_file."
```

```
echo "  -w text      Запись в файл строки text."
```

```
echo "  -r          Удаление файла test_file."
```

```
echo "  -h          Справка."
```

```
echo
```

```
}
```

```
function create_file() {
```

```
touch test_file
```

```
}
```

```
function write_to_file {
```

```
echo "$TEXT" >> test_file
```

```
}
```

```
function remove_file {
```

```
rm test_file
```

```
}
```

```
# Если скрипт запущен без аргументов, открываем справку.
```

```
if [ $# = 0 ]; then
```

```
    print_help
```

```
fi
```

```
while getopts ":cw:r" opt ;
```

```
do
```

```
    case $opt in
```

```
        c) create_file;
```

```
        ;;
```

```
        w) TEXT=$OPTARG;
```

```
        write_to_file
```

```
        ;;
```

```
        r) remove_file
```

```
        ;;
```

```
        *) echo "Неправильный параметр";
```

```
echo "Для вызова справки запустите $ME -h";
```

```
exit 1
```

```
;;
```

```
esac
```

```
done
```

Итак, что мы имеем?

- Команда **set -e** остановит скрипт, если при его выполнении возникнет ошибка (подробнее о других опциях [здесь](#)).
- Основные операции скрипта упакованы в функции. Конечно, глупо помещать в функции по одной команде, но это лишь для примера, в реальности их может быть ну очень много.
- Функция **getopts** разбирает переданные аргументы. За ней перечисляются допустимые опции. Двоеточие после опции 'w' означает, что с данной опцией идет дополнительный аргумент, который помещается в переменную **\$OPTARG**.
- Опции можно комбинировать, но стоит учитывать то, что они выполняются по порядку. Это значит, что если мы выполним **script -rc** то сначала файл будет удален, а затем создан. При этом, если файла не существовало, то скрипт завершится с ошибкой, не дойдя до создания файла.
- Также стоит учитывать то, что после ключа 'w' обязательно должен следовать аргумент. Если он будет отсутствовать, то скрипт выполнит опцию '*' (по умолчанию). Интересно, что если запустить **script -wr Hallo**, то опция 'r' будет воспринята как дополнительный параметр к опции 'w', а 'Hallo' проигнорировано. Правильно будет **script -w Hallo -r**

Подробнее о **getopts** можно узнать [здесь](#).

Выбор

В предыдущей статье я рассматривал выбор варианта выполнения с помощью **case**. А сейчас рассмотрим создание меню с помощью конструкции **select**, которая позволяет создать простые нумерованные меню.

```
#!/bin/bash
```

```
# Изменение строки приветствия
```

```
PS3='Выберите операционную систему: '
```

```
select OS in "Linux" "Windows" "Mac OS" "BolgenOS"
```

```
do
```

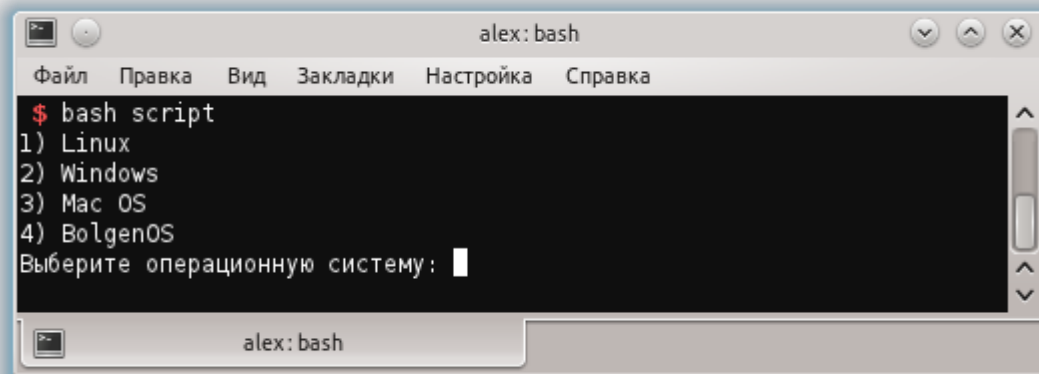
```
echo
```

```
echo "Вы выбрали $OS!"
```

```
echo
```

```
break
```

```
done
```



Подробнее описано [здесь](#).

Логирование

Бывает удобно не выводить сообщения на экран, а записывать их в лог-файл. Особенно если скрипт запускается при старте системы. Для этого можно использовать обычную запись в файл.

```
#!/bin/bash
```

```
NAME=`basename $0`
```

```
TIME=`date +%F\ %H:%M:%S`
```

```
TYPE='<info>'
```

```
echo "$TIME $NAME: $TYPE Operation completed successfully" >>
```

```
/tmp/log
```

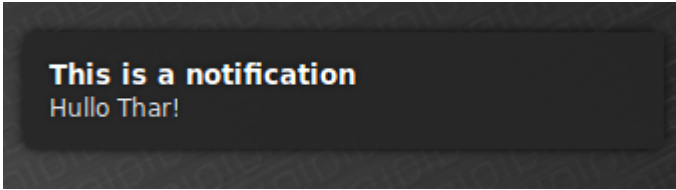
Но есть и специальный инструмент ведения логов — **logger**.

```
logger Operation completed successfully
```

```
sudo tail /var/log/syslog
```

Подробнее [тут](#).

Оповещения на рабочем столе



This is a notification
Hullo Thar!

Немножко развлечемся и поиграем с нотиферами. Для начала поставим нужный пакет:

```
sudo apt-get install libnotify-bin
```

Теперь выполним простейший пример прямо в терминале:

```
notify-send --expire-time=10000 "Привет" "Я слежу за тобой"
```

Об этом уже писали на [Хабре](#).

Клавиатурные индикаторы

Хотите поморгать лампочками на клавиатуре? Да, пожалуйста!

```
#!/bin/bash
```

```
setleds -D +caps < /dev/tty7
```

```
sleep 1
```

```
setleds -D -caps < /dev/tty7
```

Скрипт необходимо запускать с правами рута!

Звуковые сигналы

Звуковые сигналы можно подавать несколькими способами:

- С помощью управляющей последовательности на системный динамик

- `echo -e "\a"`

- С помощью утилиты **beep**

- `beep 659 120`

- Консольными плеерами, например `aplay`, `mplayer` и т.д.
- Синтезатором речи.

Первые два способа у меня не сработали, скорее всего из-за настроек терминала.

Открытие/закрытие сидиромы

```
#!/bin/bash
```

```
# открыть сидиром
```

```
eject
```

```
# закрыть сидиром
```

```
eject -t
```