Bash-скрипты, часть 5: сигналы, фоновые задачи, управление сценариями

<https://likegeeks.com/linux-bash-scripting-awesome-guide-part5/>

* [Блог компании RUVDS.com](https://habr.com/company/ruvds/),
* [Настройка Linux](https://habr.com/hub/linux/),
* [Серверное администрирование](https://habr.com/hub/s_admin/)
* Перевод

[Bash-скрипты: начало](https://habrahabr.ru/company/ruvds/blog/325522/)  
[Bash-скрипты, часть 2: циклы](https://habrahabr.ru/company/ruvds/blog/325928/)  
[Bash-скрипты, часть 3: параметры и ключи командной строки](https://habrahabr.ru/company/ruvds/blog/326328/)   
[Bash-скрипты, часть 4: ввод и вывод](https://habrahabr.ru/company/ruvds/blog/326594/)   
[Bash-скрипты, часть 5: сигналы, фоновые задачи, управление сценариями](https://habrahabr.ru/company/ruvds/blog/326826/)  
[Bash-скрипты, часть 6: функции и разработка библиотек](https://habrahabr.ru/company/ruvds/blog/327248/)  
[Bash-скрипты, часть 7: sed и обработка текстов](https://habrahabr.ru/company/ruvds/blog/327530/)  
[Bash-скрипты, часть 8: язык обработки данных awk](https://habrahabr.ru/company/ruvds/blog/327754/)  
[Bash-скрипты, часть 9: регулярные выражения](https://habrahabr.ru/company/ruvds/blog/327896/)  
[Bash-скрипты, часть 10: практические примеры](https://habrahabr.ru/company/ruvds/blog/328346/)  
[Bash-скрипты, часть 11: expect и автоматизация интерактивных утилит](https://habrahabr.ru/company/ruvds/blog/328436/)

[](https://habrahabr.ru/company/ruvds/blog/326826/)  
  
В прошлый раз мы говорили о работе с потоками ввода, вывода и ошибок в bash-скриптах, о дескрипторах файлов и о перенаправлении потоков. Сейчас вы знаете уже достаточно много для того, чтобы писать что-то своё. На данном этапе освоения bash у вас вполне могут возникнуть вопросы о том, как управлять работающими скриптами, как автоматизировать их запуск.  
  
[https://habrastorage.org/files/1ba/550/d25/1ba550d25e8846ce8805de564da6aa63.png](https://ruvds.com/ru-rub/#order)  
  
До сих пор мы вводили имена скриптов в командную строку и нажимали Enter, что приводило к немедленному запуску программ, но это — не единственный способ вызова сценариев. Сегодня мы поговорим о том как скрипт может работать с сигналами Linux, о различных подходах к запуску скриптов и к управлению ими во время работы.

Сигналы Linux

В Linux существует более трёх десятков сигналов, которые генерирует система или приложения. Вот список наиболее часто используемых, которые наверняка пригодятся при разработке сценариев командной строки.

|  |  |  |
| --- | --- | --- |
| **Код сигнала** | **Название** | **Описание** |
| 1 | SIGHUP | Закрытие терминала |
| 2 | SIGINT | Сигнал остановки процесса пользователем с терминала (CTRL + C) |
| 3 | SIGQUIT | Сигнал остановки процесса пользователем с терминала (CTRL + \) с дампом памяти |
| 9 | SIGKILL | Безусловное завершение процесса |
| 15 | SIGTERM | Сигнал запроса завершения процесса |
| 17 | SIGSTOP | Принудительная приостановка выполнения процесса, но не завершение его работы |
| 18 | SIGTSTP | Приостановка процесса с терминала (CTRL + Z), но не завершение работы |
| 19 | SIGCONT | Продолжение выполнения ранее остановленного процесса |

Если оболочка bash получает сигнал SIGHUP когда вы закрываете терминал, она завершает работу. Перед выходом она отправляет сигнал SIGHUP всем запущенным в ней процессам, включая выполняющиеся скрипты.  
  
Сигнал SIGINT приводит к временной остановке работы. Ядро Linux перестаёт выделять оболочке процессорное время. Когда это происходит, оболочка уведомляет процессы, отправляя им сигнал SIGINT.  
  
Bash-скрипты не контролируют эти сигналы, но они могут распознавать их и выполнять некие команды для подготовки скрипта к последствиям, вызываемым сигналами.

Отправка сигналов скриптам

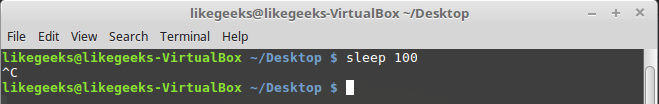
Оболочка bash позволяет вам отправлять скриптам сигналы, пользуясь комбинациями клавиш на клавиатуре. Это оказывается очень кстати если нужно временно остановить выполняющийся скрипт или завершить его работу.

Завершение работы процесса

Комбинация клавиш CTRL + C генерирует сигнал SIGINT и отправляет его всем процессам, выполняющимся в оболочке, что приводит к завершению их работы.  
  
Выполним в оболочке такую команду:

$ sleep 100

После этого завершим её работу комбинацией клавиш CTRL + C.



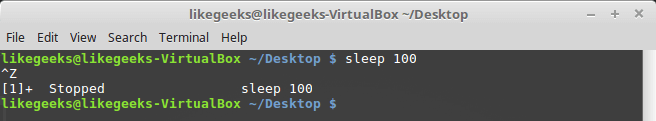
*Завершение работы процесса с клавиатуры*

Временная остановка процесса

Комбинация клавиш CTRL + Z позволяет сгенерировать сигнал SIGTSTP, который приостанавливает работу процесса, но не завершает его выполнение. Такой процесс остаётся в памяти, его работу можно возобновить. Выполним в оболочке команду:

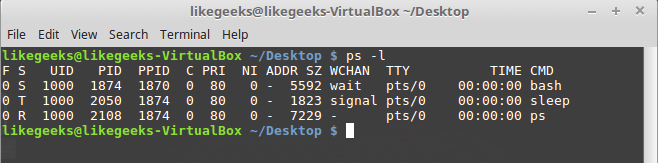
$ sleep 100

И временно остановим её комбинацией клавиш CTRL + Z.



*Приостановка процесса*  
  
Число в квадратных скобках — это номер задания, который оболочка назначает процессу. Оболочка рассматривает процессы, выполняющиеся в ней, как задания с уникальными номерами. Первому процессу назначается номер 1, второму — 2, и так далее.  
  
Если вы приостановите задание, привязанное к оболочке, и попытаетесь выйти из неё, bash выдаст предупреждение.  
  
Просмотреть приостановленные задания можно такой командой:

ps –l



*Список заданий*  
  
В колонке S, выводящей состояние процесса, для приостановленных процессов выводится T. Это указывает на то, что команда либо приостановлена, либо находится в состоянии трассировки.  
  
Если нужно завершить работу приостановленного процесса, можно воспользоваться командой kill. Подробности о ней можно почитать [здесь](https://likegeeks.com/main-linux-commands-easy-guide/).  
  
Выглядит её вызов так:

kill processID

Перехват сигналов

Для того, чтобы включить в скрипте отслеживание сигналов Linux, используется команда trap. Если скрипт получает сигнал, указанный при вызове этой команды, он обрабатывает его самостоятельно, при этом оболочка такой сигнал обрабатывать не будет.  
  
Команда trap позволяет скрипту реагировать на сигналы, в противном случае их обработка выполняется оболочкой без его участия.  
  
Рассмотрим пример, в котором показано, как при вызове команды trap задаётся код, который надо выполнить, и список сигналов, разделённых пробелами, которые мы хотим перехватить. В данном случае это всего один сигнал:

#!/bin/bash

trap "echo ' Trapped Ctrl-C'" SIGINT

echo This is a test script

count=1

while [ $count -le 10 ]

do

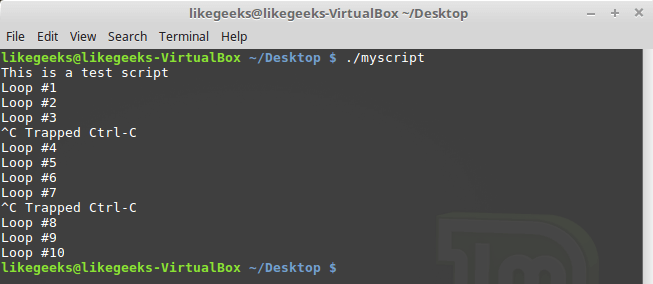
echo "Loop #$count"

sleep 1

count=$(( $count + 1 ))

done

Команда trap, использованная в этом примере, выводит текстовое сообщение всякий раз, когда она обнаруживает сигнал SIGINT, который можно сгенерировать, нажав Ctrl + C на клавиатуре.



*Перехват сигналов*  
  
Каждый раз, когда вы нажимаете клавиши CTRL + C, скрипт выполняет команду echo, указанную при вызове trace вместо того, чтобы позволить оболочке завершит его работу.

Перехват сигнала выхода из скрипта

Перехватить сигнал выхода из скрипта можно, использовав при вызове команды trap имя сигнала EXIT:

#!/bin/bash

trap "echo Goodbye..." EXIT

count=1

while [ $count -le 5 ]

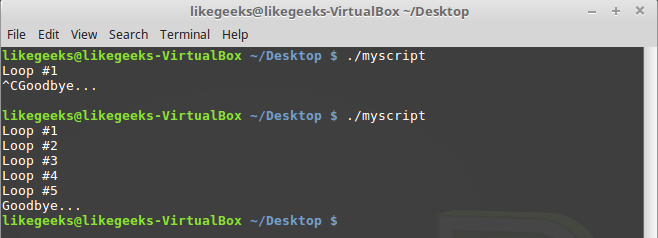
do

echo "Loop #$count"

sleep 1

count=$(( $count + 1 ))

done



*Перехват сигнала выхода из скрипта*  
  
При выходе из скрипта, будь то нормальное завершение его работы или завершение, вызванное сигналом SIGINT, сработает перехват и оболочка исполнит команду echo.

Модификация перехваченных сигналов и отмена перехвата

Для модификации перехваченных скриптом сигналов можно выполнить команду trap с новыми параметрами:

#!/bin/bash

trap "echo 'Ctrl-C is trapped.'" SIGINT

count=1

while [ $count -le 5 ]

do

echo "Loop #$count"

sleep 1

count=$(( $count + 1 ))

done

trap "echo ' I modified the trap!'" SIGINT

count=1

while [ $count -le 5 ]

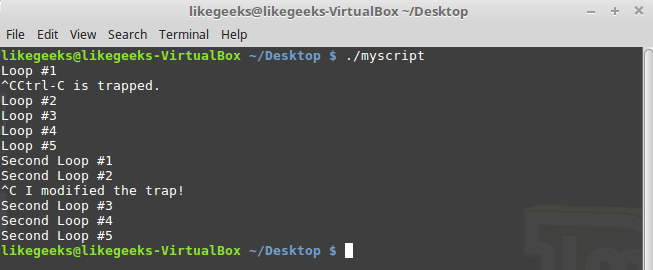
do

echo "Second Loop #$count"

sleep 1

count=$(( $count + 1 ))

done



*Модификация перехвата сигналов*  
  
После модификации сигналы будут обрабатываться по-новому.  
  
Перехват сигналов можно и отменить, для этого достаточно выполнить команду trap, передав ей двойное тире и имя сигнала:

#!/bin/bash

trap "echo 'Ctrl-C is trapped.'" SIGINT

count=1

while [ $count -le 5 ]

do

echo "Loop #$count"

sleep 1

count=$(( $count + 1 ))

done

trap -- SIGINT

echo "I just removed the trap"

count=1

while [ $count -le 5 ]

do

echo "Second Loop #$count"

sleep 1

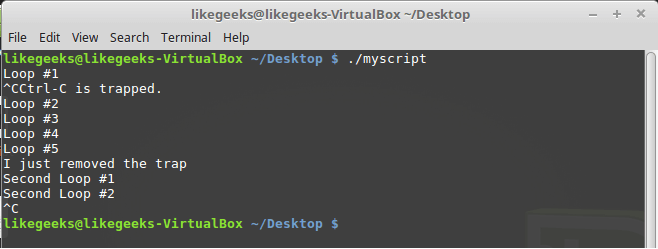
count=$(( $count + 1 ))

done

Если скрипт получит сигнал до отмены перехвата, он обработает его так, как задано в действующей команде trap. Запустим скрипт:

$ ./myscript

И нажмём CTRL + C на клавиатуре.



*Сигнал, перехваченный до отмены перехвата*  
  
Первое нажатие CTRL + C пришлось на момент исполнения скрипта, когда перехват сигнала был в силе, поэтому скрипт исполнил назначенную сигналу команду echo. После того, как исполнение дошло до команды отмены перехвата, команда CTRL + C сработала обычным образом, завершив работу скрипта.

Выполнение сценариев командной строки в фоновом режиме

Иногда bash-скриптам требуется немало времени для выполнения некоей задачи. При этом вам может понадобиться возможность нормально работать в командной строке, не дожидаясь завершения скрипта. Реализовать это не так уж и сложно.  
  
Если вы видели список процессов, выводимый командой ps, вы могли заметить процессы, которые выполняются в фоне и не привязаны к терминалу.  
Напишем такой скрипт:

#!/bin/bash

count=1

while [ $count -le 10 ]

do

sleep 1

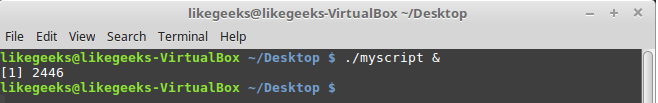
count=$(( $count + 1 ))

done

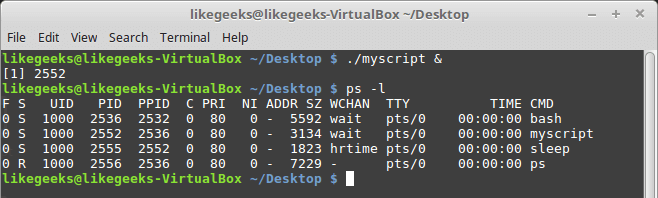
Запустим его, указав после имени символ амперсанда (&):

$ ./myscipt &

Это приведёт к тому, что он будет запущен как фоновый процесс.



*Запуск скрипта в фоновом режиме*  
  
Скрипт будет запущен в фоновом процессе, в терминал выведется его идентификатор, а когда его выполнение завершится, вы увидите сообщение об этом.  
  
Обратите внимание на то, что хотя скрипт выполняется в фоне, он продолжает использовать терминал для вывода сообщений в STDOUT и STDERR, то есть, выводимый им текст или сообщения об ошибках можно будет увидеть в терминале.



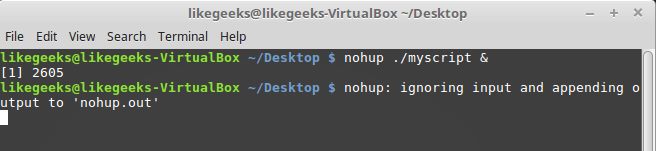
*Список процессов*  
  
При таком подходе, если выйти из терминала, скрипт, выполняющийся в фоне, так же завершит работу.  
  
Что если нужно, чтобы скрипт продолжал работать и после закрытия терминала?

Выполнение скриптов, не завершающих работу при закрытии терминала

Скрипты можно выполнять в фоновых процессах даже после выхода из терминальной сессии. Для этого можно воспользоваться командой nohup. Эта команда позволяет запустить программу, блокируя сигналы SIGHUP, отправляемые процессу. В результате процесс будет исполняться даже при выходе из терминала, в котором он был запущен.  
  
Применим эту методику при запуске нашего скрипта:

nohup ./myscript &

Вот что будет выведено в терминал.



*Команда nohup*  
  
Команда nohup отвязывает процесс от терминала. Это означает, что процесс потеряет ссылки на STDOUTи STDERR. Для того, чтобы не потерять данные, выводимые скриптом, nohup автоматически перенаправляет сообщения, поступающие в STDOUT и в STDERR, в файл nohup.out.  
  
Обратите внимание на то, что при запуске нескольких скриптов из одной и той же директории то, что они выводят, попадёт в один файл nohup.out.

Просмотр заданий

Команда jobs позволяет просматривать текущие задания, которые выполняются в оболочке. Напишем такой скрипт:

#!/bin/bash

count=1

while [ $count -le 10 ]

do

echo "Loop #$count"

sleep 10

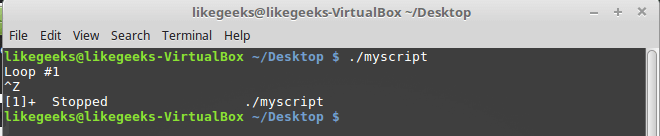
count=$(( $count + 1 ))

done

Запустим его:

$ ./myscript

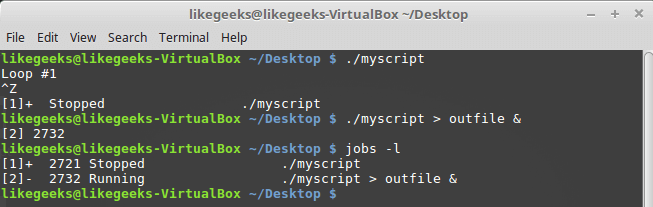
И временно остановим комбинацией клавиш CTRL + Z.



*Запуск и приостановка скрипта*  
  
Запустим тот же скрипт в фоновом режиме, при этом перенаправим вывод скрипта в файл так, чтобы он ничего не выводил на экране:

$ ./myscript > outfile &

Выполнив теперь команду jobs, мы увидим сведения как о приостановленном скрипте, так и о том, который работает в фоне.



*Получение сведений о скриптах*  
  
Ключ -l при вызове команды jobs указывает на то, что нам нужны сведения об ID процессов.

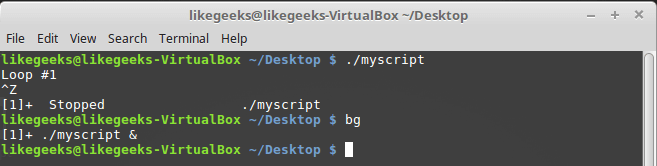
Перезапуск приостановленных заданий

Для того, чтобы перезапустить скрипт в фоновом режиме, можно воспользоваться командой bg.  
  
Запустим скрипт:

$ ./myscript

Нажмём CTRL + Z, что временно остановит его выполнение. Выполним следующую команду:

$ bg



*Команда bg*  
  
Теперь скрипт выполняется в фоновом режиме.  
  
Если у вас имеется несколько приостановленных заданий, для перезапуска конкретного задания команде bg можно передать его номер.  
  
Для перезапуска задания в обычном режиме воспользуйтесь командой fg:

$ fg 1

Планирование запуска скриптов

Linux предоставляет пару способов запуска bash-скриптов в заданное время. Это команда at и планировщик заданий cron.  
  
Вызов команды at выглядит так:

at [-f filename] time

Эта команда распознаёт множество форматов указания времени.

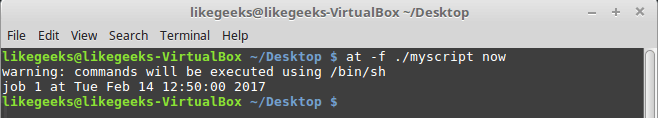
* Стандартный, с указанием часов и минут, например — 10:15.
* С использованием индикаторов AM/PM, до или после полудня, например — 10:15PM.
* С использованием специальных имён, таких, как now, noon, midnight.

В дополнение к возможности указания времени запуска задания, команде at можно передать и дату, используя один из поддерживаемых ей форматов.

* Стандартный формат указания даты, при котором дата записывается по шаблонам MMDDYY, MM/DD/YY, или DD.MM.YY.
* Текстовое представление даты, например, Jul 4 или Dec 25, при этом год можно указать, а можно обойтись и без него.
* Запись вида now + 25 minutes.
* Запись вида 10:15PM tomorrow.
* Запись вида 10:15 + 7 days.

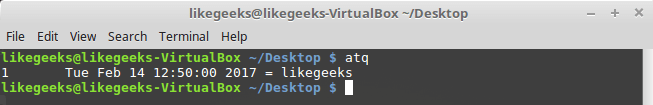
Не будем углубляться в эту тему, рассмотрим простой вариант использования команды:

$ at -f ./myscript now



*Планирование заданий с использованием команды at*  
  
Ключ -M при вызове at используется для отправки того, что выведет скрипт, по электронной почте, если система соответствующим образом настроена. Если отправка электронного письма невозможна, этот ключ просто подавит вывод.  
  
Для того чтобы посмотреть список заданий, ожидающих выполнения, можно воспользоваться командой atq:

$ atq

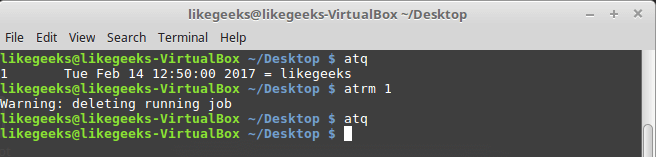


*Список заданий, ожидающих выполнения*

Удаление заданий, ожидающих выполнения

Удалить задание, ожидающее выполнения, позволяет команда atrm. При её вызове указывают номер задания:

$ atrm 18



*Удаление задания*

Запуск скриптов по расписанию

Планирование однократного запуска скриптов с использованием команды at способно облегчить жизнь во многих ситуациях. Но как быть, если нужно, чтобы скрипт выполнялся в одно и то же время ежедневно, или раз в неделю, или раз в месяц?  
  
В Linux имеется утилита crontab, позволяющая планировать запуск скриптов, которые нужно выполнять регулярно.  
  
Crontab выполняется в фоне и, основываясь на данных в так называемых cron-таблицах, запускает задания по расписанию.  
  
Для того, чтобы просмотреть существующую таблицу заданий cron, воспользуйтесь такой командой:

$ crontab –l

При планировании запуска скрипта по расписанию crontab принимает данные о том, когда нужно выполнить задание, в таком формате:

минута, час, день месяца, месяц, день недели.

Например, если надо, чтобы некий скрипт с именем command выполнялся ежедневно в 10:30, этому будет соответствовать такая запись в таблице заданий:

30 10 \* \* \* command

Здесь универсальный символ «\*», использованный для полей, задающих день месяца, месяц и день недели, указывает на то, что cron должен выполнять команду каждый день каждого месяца в 10:30.  
  
Если, например, надо, чтобы скрипт запускался в 4:30PM каждый понедельник, понадобится создать в таблице заданий такую запись:

30 16 \* \* 1 command

Нумерация дней недели начинается с 0, 0 означает воскресенье, 6 — субботу. Вот ещё один пример. Здесь команда будет выполняться в 12 часов дня в первый день каждого месяца.

00 12 1 \* \* command

Нумерация месяцев начинается с 1.  
Для того чтобы добавить запись в таблицу, нужно вызвать crontab с ключом -e:

crontab –e

Затем можно вводить команды формирования расписания:

30 10 \* \* \* /home/likegeeks/Desktop/myscript

Благодаря этой команде скрипт будет вызываться ежедневно в 10:30. Если вы столкнётесь с ошибкой «Resource temporarily unavailable», выполните нижеприведённую команду с правами root-пользователя:

$ rm -f /var/run/crond.pid

Организовать периодический запуск скриптов с использованием cron можно ещё проще, воспользовавшись несколькими специальными директориями:

/etc/cron.hourly

/etc/cron.daily

/etc/cron.weekly

/etc/cron.monthly

Если поместить файл скрипта в одну из них, это приведёт, соответственно, к его ежечасному, ежедневному, еженедельному или ежемесячному запуску.

Запуск скриптов при входе в систему и при запуске оболочки

Автоматизировать запуск скриптов можно, опираясь на различные события, такие, как вход пользователя в систему или запуск оболочки. [Тут](https://likegeeks.com/linux-environment-variables/) можно почитать о файлах, которые обрабатываются в подобных ситуациях. Например, это следующие файлы:

$HOME/.bash\_profile

$HOME/.bash\_login

$HOME/.profile

Для того, чтобы запускать скрипт при входе в систему, поместите его вызов в файл .bash\_profile.  
  
А как насчёт запуска скриптов при открытии терминала? Организовать это поможет файл .bashrc.

Итоги

Сегодня мы разобрали вопросы, касающиеся управления жизненным циклом сценариев, поговорили о том, как запускать скрипты в фоне, как планировать их выполнение по расписанию. В следующий раз читайте о функциях в bash-скриптах и о разработке библиотек.  
  
Уважаемые читатели! А вы пользуетесь средствами планирования запуска сценариев командной строки по расписанию? Если да — расскажите пожалуйста о них.