Небольшое руководство по getopts

Описание

Обратите внимание, что `getopts` он не способен анализировать длинные параметры в стиле GNU (`--myoption`) или длинные параметры в стиле XF86 (`-myoption`) . Поэтому, если вы хотите профессионально анализировать аргументы командной строки, `getopts` это может работать, а может и не работать для вас. В отличие от своего старшего брата `getopt` (обратите внимание на пропущенные буквы *s*!), Это встроенная команда оболочки. Преимущества заключаются в:

- Нет необходимости передавать позиционные параметры во внешнюю программу.
- Будучи встроенным, `getopts` может устанавливать переменные оболочки для использования для синтаксического анализа (невозможно для *внешнего* процесса!)
- Нет необходимости спорить с несколькими `getopt` реализациями, в которых в прошлом были глючные концепции (пробелы, ...)
- `getopts` определяется в POSIX®.

Некоторые другие методы анализа позиционных параметров - не использующие ни **getopt**, ни **getopts** - описаны в: Как обрабатывать позиционные параметры .

Терминология

Полезно знать, о чем мы здесь говорим, так что давайте посмотрим… Рассмотрим следующую командную строку:

```
mybackup -x -f /etc/mybackup.conf -r ./foo.txt ./bar.txt
```

Все это позиционные параметры, но их можно разделить на несколько логических групп:

- `-x` это **опция** (она же **флаг** или **переключатель**). Он состоит из тире (`-`), за которым следует **один** символ.
- `-f` также является опцией, но у этой опции есть связанный аргумент **опции (аргумент** для опции `-f`): `/etc/mybackup.conf` . Аргумент опции обычно является аргументом, следующим за самой опцией, но это не обязательно.

Объединение параметра и аргумента параметра в один аргумент `-f/etc/mybackup.conf` допустимо.

- `-r` зависит от конфигурации. В этом примере `-r` не принимает аргументы, поэтому это автономный вариант, например `-x` .
- `./foo.txt и ./bar.txt` являются оставшимися аргументами без каких-либо связанных параметров. Они часто используются в качестве **массовых аргументов**. Например, имена файлов, указанные для `cp(1)` , или аргументы, для которых не требуется распознавание параметра из-за предполагаемого поведения программы. POSIX® называет их **операндами**.

Чтобы дать вам представление о том, почему `getopts` это полезно, приведенная выше командная строка эквивалентна:

```
mybackup -xrf /etc/mybackup.conf ./foo.txt ./bar.txt
```

который сложно разобрать без помощи `getopts` .

Флагами опций могут быть символы **верхнего и нижнего регистра** или **цифры**. Он может распознавать другие символы, но это не рекомендуется (удобство использования и, возможно, проблемы со специальными символами).

Как это работает

В общем, вам нужно позвонить `getopts` несколько раз. Каждый раз он будет использовать следующий позиционный параметр и возможный аргумент, если он поддается анализу, и предоставлять его вам. `getopts` не изменит набор позиционных параметров. Если вы хотите изменить их, это должно быть сделано вручную:

```
сдвиг $((OPTIND-1))
# теперь сделайте что-нибудь с $@
```

Поскольку `getopts` устанавливает статус выхода *FALSE*, когда ничего не осталось для анализа, его легко использовать в цикле while:

```
в то время как getopts ...; делать
 ...
Выполнено
```

`getopts` проанализирует параметры и их возможные аргументы. Он остановит синтаксический анализ при первом аргументе, не являющемся параметром (строка, которая не начинается с дефиса (`-`), который не является аргументом для любого параметра перед ним). Он также прекратит синтаксический анализ, когда увидит `--` (двойной дефис), что означает конец опций.

Используемые переменные

переменная	Описание

переменная	Описание
OPTIND	Содержит индекс следующего обрабатываемого аргумента. Вот как `getopts` "запоминает" свой собственный статус между вызовами. Также полезно для изменения позиционных параметров после обработки `getopts` . `OPTIND` изначально установлено значение 1, и его **необходимо переустановить на 1, если вы хотите снова проанализировать что-либо с помощью getopts**
OPTARG	Эта переменная устанавливается в любой аргумент для параметра, найденного `getopts` . Он также содержит флаг параметра неизвестной опции.
OPTERR	(Значения 0 или 1) Указывает, должен ли Bash отображать сообщения об ошибках, сгенерированные `getopts` встроенным. Значение инициализируется равным **1** при каждом запуске командной оболочки, поэтому обязательно всегда устанавливайте его равным **0**, если не хотите видеть раздражающие сообщения! **`OPTERR` не определяется POSIX для `getopts` встроенной утилиты — только для функции C `getopt()` в `unistd.h` (`opterr`).** `OPTERR` зависит от bash и не поддерживается такими оболочками, как ksh93, mksh, zsh или dash.

`getopts` также использует эти переменные для отчетов об ошибках (для них установлены комбинации значений, которые невозможны при нормальной работе).

Укажите, что вы хотите

Базовый синтаксис для `getopts` :

```
getopts OPTSTRING VARNAME [АРГУМЕНТЫ ...]
```

где:

`OPTSTRING`	указывает `getopts` , какие параметры следует ожидать и где ожидать аргументы (см. Ниже)
`VARNAME`	указывает `getopts` , какую переменную оболочки использовать для создания отчетов о параметрах
`ARGS`	указывает `getopts` на синтаксический анализ этих необязательных слов вместо позиционных параметров

Опция-строка

Строка параметров указывает `getopts` , какие параметры следует ожидать и какие из них должны иметь аргумент. Синтаксис очень прост — каждый символ опции просто называется как есть, в этом примере-строке будет указано `getopts` , что искать `-f` , `-A` и `-x` :

```
имя ПЕРЕМЕННОЙ для факса getopts
```

Если вы хотите `getopts` ожидать аргумент для опции, просто поместите `:` (двоеточие) после соответствующего флага опции. Если вы хотите `-A` ожидать аргумента (т. Е. Стать `-A SOMETHING`), просто сделайте:

```
getopts fA: x VARNAME
```

Если **самым первым символом** строки параметров является `:` (двоеточие), что обычно было бы бессмысленно, потому что перед ним нет буквы параметра, `getopts` переключается в "**режим автоматического сообщения об ошибках**". В продуктивных сценариях это обычно то, что вам нужно, потому что это позволяет вам самостоятельно обрабатывать ошибки, не беспокоясь о раздражающих сообщениях.

Пользовательские аргументы для анализа

`getopts` Утилита анализирует позиционные параметры текущей оболочки или функции по умолчанию (что означает, что она анализирует `"$@"`).

Вы можете предоставить свой собственный набор аргументов утилите для синтаксического анализа. Всякий раз, когда после `VARNAME` параметра задаются дополнительные аргументы, `getopts` он пытается проанализировать не позиционные параметры, а эти заданные слова.

Таким образом, вы можете проанализировать любой набор параметров, который вам нравится, например, из массива:

```
в то время как getopts:f:h выбирает "${MY_OWN_SET[@]}"; делать
 ...
Выполнено
```

Вызов `getopts` **без** этих дополнительных аргументов **эквивалентен** явному вызову его с помощью `"$@"` :

```
getopts ... "$ @"
```

Сообщение об ошибках

Что касается отчетов об ошибках, есть два режима `getopts` , в которых можно запускать:

- подробный режим
- тихий режим

Для продуктивных скриптов я рекомендую использовать тихий режим, поскольку все выглядит более профессионально, когда вы не видите раздражающих стандартных сообщений. Кроме того, с ним проще обращаться, поскольку случаи сбоев указаны более простым способом.

Подробный режим

недопустимый вариант	`VARNAME` установлено значение `?` (вопросительный знак) и `OPTARG` не установлено

требуемый аргумент не найден	`VARNAME` установлено значение `?` (вопросительный знак), `OPTARG` не установлено и *выводится сообщение об ошибке*

Тихий режим

недопустимый вариант	`VARNAME` имеет значение `?` (вопросительный знак) и `OPTARG` имеет значение (недопустимый) символ опции
требуемый аргумент не найден	`VARNAME` имеет значение `:` (двоеточие) и `OPTARG` содержит соответствующий символ параметра

Используя его

Первый пример

Достаточно сказано - действуй!

Давайте поиграем с очень простым случаем: ожидается только один вариант (`-a`), без каких-либо аргументов. Также мы отключаем *подробную обработку ошибок*, предваряя всю строку параметров двоеточием (`:`):

```
": a"

getopts  while #!/bin/bash opt; сделайте
   case $opt  в
a)
       echo "-a был запущен!" > &2
      ;;
 \?)
       эхо  "Недопустимый вариант: -$ OPTARG" > & 2
      ;;
  esac
 выполнено
```

Я поместил это в файл `go_test.sh` с именем, которое вы увидите ниже в примерах.

Давайте проведем несколько тестов:

Вызывая его без каких-либо аргументов

```
$ ./go_test.sh
$
```

Ничего не произошло? Правильно. `getopts` не видел никаких допустимых или недопустимых параметров (буквам предшествует тире), поэтому он не был запущен.

Вызывая его с аргументами, не являющимися параметрами

```
$ ./go_test.sh /etc/passwd
$
```

Опять же — ничего не произошло. Тот **же самый** случай: `getopts` не видел никаких допустимых или недопустимых параметров (буквам предшествует тире), поэтому он не был запущен.

Аргументы, приведенные в вашем скрипте, конечно, доступны как `$1` - `${N}` .

Вызывая его с помощью параметров-аргументов

Теперь давайте запустим `getopts` : укажите параметры.

Во-первых, **недопустимый**:

```
$ ./go_test.sh -б
Недопустимый вариант: -b
$
```

Как и ожидалось, `getopts` я не принял эту опцию и поступил так, как указано выше: он поместил `?` в `$opt` и недопустимый символ опции (`b`) `$OPTARG` . С помощью нашего `case` заявления мы смогли это обнаружить.

Теперь **действительный** (`-a`):

```
$ ./go_test.sh -a
-a был запущен!
$
```

Как видите, обнаружение работает отлично. Значение `a` было введено в переменную `$opt` для нашего описания случая.

Конечно, при вызове можно **смешивать допустимые и недопустимые** параметры:

```
$ ./go_test.sh -a -x -b -c
-a был запущен!
Недопустимый вариант: -x
Недопустимый вариант: -b
Недопустимый вариант: -c
$
```

Наконец, конечно, возможно предоставить наш вариант **несколько раз**:

```
$ ./go_test.sh -a -a -a -a
-a был запущен!
-a был запущен!
-a был запущен!
-a был запущен!
$
```

Последние примеры подводят нас к некоторым моментам, которые вы можете рассмотреть:

- **недопустимые параметры не останавливают обработку**: если вы хотите остановить скрипт, вы должны сделать это самостоятельно (`exit` в нужном

месте)
- **возможно несколько идентичных опций**: если вы хотите запретить их, вам нужно проверить вручную (например, установив переменную или около того)

Вариант с аргументом

Давайте расширим наш пример, приведенный выше. Совсем немного:

- `-a` теперь принимает аргумент
- при ошибке синтаксический анализ завершается с `exit 1`

```
": a:"

getopts  while #!/bin/bash opt; do
   case $opt  in
a)
      echo "-был запущен параметр: $OPTARG" > &2
      ;;
 \?)
      echo "Недопустимый вариант: -$OPTARG" > &2
       выход  1
      ;;
 :)
      echo "Option -$OPTARG требует аргумента". >&2
       выход  1
      ;;
  esac
 сделано
```

Давайте проведем те же тесты, что и в предыдущем примере:

Вызывая его без каких-либо аргументов

```
$ ./go_test.sh
$
```

Как и выше, ничего не произошло. Он не был запущен.

Вызывая его с аргументами, не являющимися параметрами

```
$ ./go_test.sh /etc/passwd
$
```

Тот **же самый** случай: он не был запущен.

Вызывая его с помощью параметров-аргументов

Недопустимый вариант:

```
$ ./go_test.sh -б
Недопустимый вариант: -b
$
```

Как и ожидалось, как указано выше, `getopts` не принял этот вариант и действовал как запрограммированный.

Допустимый вариант, но без обязательного **аргумента**:

```
$ ./go_test.sh -a
Опция -a требует аргумента.
$
```

Вариант был в порядке, но не хватает аргумента.

Давайте приведем **аргумент**:

```
$ ./go_test.sh -a /etc/passwd
-a был запущен, параметр: /etc/passwd
$
```

Смотрите также

- Внутренний: обработка позиционных параметров
- Внутренний: заявление о случае
- Внутренний: цикл while
- POSIX getopts(1) (http://pubs.opengroup.org/onlinepubs/9699919799/utilities/getopts.html#tag_20_54) и getopt(3) (http://pubs.opengroup.org/onlinepubs/9699919799/functions/getopt.html)
- разбор CLI ARGV (https://stackoverflow.com/questions/192249/how-do-i-parse-command-line-arguments-in-bash)
- обработка аргументов (параметров) командной строки для скрипта (http://mywiki.wooledge.org/BashFAQ/035)

Обсуждение

Стив Лессард, 2010/07/14 15:04 ()

Спасибо за очень хороший урок. Я многому научился из него. Однако в этом руководстве, похоже, отсутствует одна вещь, как использовать необязательный аргумент 'ARGS'. Вы дали очень краткое описание того, что делает ARGS, но не объяснили, как это работает. Что значит анализировать "слова вместо позиционных параметров"?

Ян Шампера, 07.07.2010 / 14 19:42 ()

Надеюсь, теперь это понятно, лично я никогда не использовал эту функцию. Пожалуйста, предоставьте предложения, если хотите.

Спасибо за отзыв.

Марк Шиманski, 29.07.2010 03:42 ()

Извините, если я просто что-то здесь упускаю, но что с командами `> ampersand 2` in the echo?

Ян Шампера, 2010/07/29 09:55 ()

Рекомендуется печатать сообщения об ошибках и диагностические сообщения в стандартный вывод ошибок (`STDERR`). `foo > ampersand 2` делает это.

Зенаан Харкнесс, 2010/11/04 14:34 ()

> Рекомендуется печатать сообщения об ошибках и диагностические сообщения в стандартный вывод ошибок (STDERR).

Согласен, но, пожалуйста, не делайте этого для вывода -h или –help , который является ожидаемым и указанным выводом и поэтому должен перейти в стандартный вывод.

Ян Шампера, 2010/11/04 18:12 ()

Я отчасти согласен, да. Но я также допустил эту ошибку здесь и там. Когда требуется помощь, вывод текста справки не является диагнозом или сообщением об ошибке.

Джошуа, 2010/12/05 00:06 ()

Что делать, если существует несколько вариантов, и некоторые из них требуют аргументов, а некоторые нет? Кажется, я не могу заставить его работать должным образом...

Пример)

#!/bin/bash

в то время как getopts ": a: b: cde: f: g:" выбрать; сделать

```
случай $opt в
a)
было запущено echo "-a, параметр: $OPTARG" > & 2
 ;;
b)
было запущено echo "-b, параметр: $OPTARG" > & 2
 ;;
c)
было запущено echo "-c, параметр: $OPTARG" > & 2
 ;;
 d)
было запущено echo "-d, параметр: $OPTARG" > & 2
 ;;
e)
было запущено echo "-e, параметр: $OPTARG" > & 2
 ;;
 f)
было запущено echo "-w, параметр: $OPTARG" > & 2
 ;;
g)
было запущено echo "-g, параметр: $OPTARG" > & 2
 ;;
```

```
 \?)
 echo "Недопустимый параметр: - $ OPTARG" > & 2
 выход 1
 ;;
 :)
echo "Option -$OPTARG требует аргумента". >&2
 выход 1
 ;;
esac
```

Выполнено

Вот моя проблема:

./ взломать.был запущен bash -a -b -a, параметр: -b

Разве он не должен отображать, что -a отсутствует аргумент вместо того, чтобы использовать следующий параметр в качестве параметра. Что я здесь делаю не так?

Ян Шампера, 2010/12/05 06:29 ()

Вы не делаете ничего плохого. Это похоже на то, что, когда getopts ищет аргумент, он принимает следующий.

Так ведет себя большинство известных мне программ (`tar` , текстовые утилиты, ...).

Марк, 2011/01/29 19:42 ()

Как мне сделать так, чтобы при отсутствии переданных аргументов он возвращал текст со словами "пароль без аргументов, ничего не срабатывает"?

Ян Шампера, 2011/01/29 19:50 ()

Я бы сделал это, проверив $ # перед циклом while / getopts, если применимо:

```
$# ((if == 0)); затем
 ...
fi
```

Если вам действительно нужно проверить, `getopts` найдено ли что-то для обработки, вы можете создать переменную для этой проверки:

```
options_found=0

, в то время  как getopts ":xyz" выбирает; делать
   options_found= 1
 ...
сделано

, если  ((!options_found)); затем
   повторите  "параметры не найдены"
 fi
```

сараванан субраманьян, 2014/12/04 16:56 ()

Привет, я попробовал выше - но все равно, когда я запускаю с ./tw.ksh -a -f "XXX" - он устанавливает переменную options_found=1. Я хочу, чтобы он запускался только тогда, когда у обоих вариантов есть аргументы, в противном случае exit.

в то время как getopts ": hf: F: a:" arg

сделайте

options_found=1

```
case $ arg в
```

```
a)
```

```
CMD_INP=$OPTARG
```

```
            ;;
```

```
F)
```

```
HOSTS_FILE="$STAGEDIR/$OPTARG/hosts"
```

```
если [ ! -f $HOSTS_FILE ]
```

```
затем
```

```
echo "Файл $HOSTS_FILE не существует, завершается"
```

```
выход
```

```
ещё
```

```
HOSTS_TO_FIX=`cat $HOSTS_FILE`
```

```
фи
```

```
            ;;
```

```
f)
```

```
HOSTS_TO_FIX=$OPTARG
```

```
            ;;
```

```
h | *)
```

```
использование
```

```
выход 1
```

```
            ;;
```

```
esac
```

Выполнено

Рейд, 2011/08/11 22:07 (), 2011/08/11 23:41 ()

Другой способ проверить, нашел ли он что-нибудь вообще, - это запустить отдельный оператор if прямо перед вызовом while getopts .

```
если (! getopts "abc: deh" opt); затем
повторите "Использование: параметры `basename $ 0` (-ab) (-зна
чение c) (-d) (-e) -h для справки";
 выйдите из $E_OPTERROR;
фи

в то время как etopts "abc: deh" выбирает; сделайте
case $opt в
a) сделайте что-нибудь;;
 б) сделайте еще один;;
 c) var= $OPTARG;;
 ...
 esac
выполнен
```

Марк, 2011/01/29 20:09 ()

Отличная работа, спасибо!

Как заставить его возвращать несколько аргументов в одной строке? например. привет -ab возвращает "вариант a вариант b"?

Ян Шампера, 2011/01/29 21:16 ()

Это не связано с getopts. Просто используйте переменные или echo без новых строк или подобных вещей, как вы бы поступили в таком случае и без getopts.

Андреа, 2011/05/02 14:22 ()

Привет. как я могу управлять двойным вызовом одной и той же опции? Я не хочу такой ситуации: ./script -a xxx -a xxx!

Ян Шампера, 2011/05/02 15:03 ()

См. Вопрос выше. Установите переменную, которая обрабатывает это, своего рода флаг, который устанавливается при вызове опции, и проверяет, была ли опция уже вызвана. Своего рода "щит".

```
A_WAS_SET=0
...
случай
...
 a)
если [[ $A_WAS_SET = 0 ]]; тогда
 A_WAS_SET=1
 # сделайте что-нибудь, что обрабатывает -a
else
echo "Опция -a уже использовалась".
 выйдите из 1
fi
;;
esac
...
```

Андреа, 2011/05/03 13:57 ()

Спасибо! Это работает!

Джо Вульф, 2011/06/22 20:33 ()

В примере Джошуа (из приведенного выше @ 2010/12/05 01:06) спрашивалось о разборе нескольких вариантов, где у некоторых есть обязательные аргументы, а у некоторых есть НЕОБЯЗАТЕЛЬНЫЕ аргументы. У меня есть скрипт, который я улучшаю. Для ВЫПОЛНЕНИЯ требуется аргумент '-e' (и '-i' для установки, '-r' для удаления и т. Д.). -e стабилен сам по себе. Моим улучшением было бы разрешить необязательный '-e <modifier>', чтобы функциональность была соответствующим образом изменена условно. Как мне определить строку getopts, чтобы указать, что '-e' является допустимым параметром для анализа и что у него МОЖЕТ быть аргумент??

Ян Шампера, 2011/06/23 06:42 (), 2011/06/23 06:58 ()

Привет,

попробуйте этот трюк. Когда вы обнаружите, что `OPTARG` von `-c` - это что-то, начинающееся с дефиса, затем сбросьте `OPTIND` и повторно запустите getopts (`continue` цикл while).

Код относительно небольшой, но я надеюсь, что вы поняли идею.

О, конечно, это не идеально и требует большей надежности. Это всего лишь пример.

```
#!/bin/bash


в то время как getopts: abc: opt; сделайте
case $opt в
a)
повторите "вариант a"
 ;;
b)
повторите "вариант b"
 ;;
c)
повторите "вариант c"

 если [[ $OPTARG = -* ]]; тогда
 ((OPTIND--))
 продолжить


 echo "(c) аргумент $OPTARG"
 ;;
 \?)
 эхо "WTF!"
 выход 1
 ;;
esac
выполнено
```

Рид, 08.08.2011 21:29 ()

Другой способ иметь "необязательный" аргумент - иметь как строчную, так и заглавную версию параметра, причем для одного требуется аргумент, а для другого он не требуется.

Джей, 07.07.2011 / 27 18:10 ()

Выделенный жирным шрифтом текст Что делать, если у вас есть флаг с НЕОБЯЗАТЕЛЬНЫМ аргументом; скажем, вызов может быть либо с именем пользователя, либо просто -a. Определяется только с помощью a: он жалуется, что аргумента нет. Я хочу, чтобы он использовал аргумент, если он есть, иначе используйте значение по умолчанию, определенное в другом месте.

Арвид Рекейт, 2011/10/07 10:04 ()

Встроенные getopts можно использовать для анализа длинных опций, поместив в строку optstring символ тире, за которым следует двоеточие ("getopts 'h-:'" или "getopts – '-:'"), вот пример, как это можно сделать:

http://stackoverflow.com/questions/402377/using-getopts-in-bash-shell-script-to-get-long-and-short-command-line-options/7680682#7680682 (http://stackoverflow.com/questions/402377/using-getopts-in-bash-shell-script-to-get-long-and-short-command-line-options/7680682#7680682)

Этот подход переносим в оболочку Debian Almquist ("dash").

Ян Шампера, 2011/10/08 08:09 ()

Очень хороший трюк!

Другой способ, который я мог себе представить (и я когда-нибудь попробую протестировать код), - это предварительная обработка позиционных параметров и преобразование длинных параметров в короткие параметры перед использованием getopts.

Аби Майкл, 2012/02/06 04:06 ()

Большое вам спасибо. Такой замечательный учебник!!!

Ян Сидло, 2012/03/09 22:04 ()

У меня вопрос. Как мне получить эти массовые аргументы или операнды? Когда у меня есть список входных файлов в конце. Например, так ./script -r r_arg -np input_file_1 input_file_2 или ./script -r r_arg -n -p p_arg input_file_1 input_file_2 . Я не знаю количество операндов (входных файлов) заранее, поэтому не могу использовать $ #.

Ян Шампера, 2012/03/10 10:08 ()

После `getopts` того, как это будет сделано, удалите все обработанные параметры с помощью

```
сдвиг $((OPTIND-1))
# теперь сделайте что-нибудь с $@
```

чандракант, 2012/03/30 06:11 ()

всем привет, нужна помощь!!

В моем случае мне нужны переключатели, чтобы быть строками, а не символами, например. getopt, как объяснено здесь, работает как filename -a <arga> -b <argb>, но я хотел в формате filename -name <arga> -age <argb>

если не использовать getopt, что еще можно использовать для реализации такого рода сценариев в оболочке (sh shell)

Заранее спасибо

Ян Шампера, 2012/04/07 08:09 ()

Смотрите Примеры в разделе Обработка позиционных параметров для некоторых способов, отличных от getopts.

Шашикант, 2012/04/10 12:43 ()

Очень хорошее объяснение. Это помогло мне.

Большое спасибо.

stib, 2012/07/23 06:19 ()

Это моя попытка получить необязательные аргументы. Если пользователь указывает -a -b, то getopts видит -b как $OPTARG для -a, поэтому мы возвращаем $OPTIND обратно на единицу, и в этом случае присваиваем $a значение по умолчанию, в противном случае $a получает значение $ OPTARG . Обратите внимание, что это не сработает, если пользователь использует параметры в форме -ab, только -a -b [CODE], в то время как getopts ": a: bc" optname; в случае "$ optname" в "a") echo a срабатывает;

```
a= $OPTARG;
если [[ "$OPTARG"=="-*" ]];
 затем ((OPTIND--));
 a="ПО УМОЛЧАНИЮ";
fi
```

;; "b") echo "b срабатывает"; ;; ":") echo нужно значение pal; ;; esac; готово; echo a= $a; [/CODE]

Рафаэль Ринальди, 2012/10/07 03:31 ()

Я пытался иметь возможность давать имена своим параметрам вместо использования только отдельных символов. Это было больно делать с getopts, я не знаю, не хватает ли мне чего-то или чего. Если есть кто-нибудь с такой же потребностью, вот мой обходной путь: http://stackoverflow.com/questions/402377/using-getopts-in-bash-shell-script-to-get-long-and-short-command-line-options/6946864#6946864 (http://stackoverflow.com/questions/402377/using-getopts-in-bash-shell-script-to-get-long-and-short-command-line-options/6946864#6946864)

Ян Шампера, 2012/10/13 09:30 ()

Привет,

как отмечалось выше, `getopts` он не предназначен для анализа длинных опций в стиле GNU (что вы и пытаетесь сделать).

Кевин Вайман, 2013/04/06 02:33 ()

Привет всем, так что мне на самом деле довольно сложно разобраться с getopts, case и shifting. Я понимаю, что сдвигает позиционный параметр и перемещается влево, присваивая ему значение [1)]. Самая большая область, с которой я борюсь, - это использование функций с getopts / case. У меня есть свои функции и их логика, определенные выше моего объявления getopts, и я хочу назначить каждому переключателю (опции) вызов (или набор) функций (функций). Кроме того, у меня есть логика одной функции, которая вызывается переключателем (мы будем использовать -m) только тогда, когда присутствует отдельный переключатель (-i), который успешно завершается первым.

Разбивка: [code] variable= "$ 2" usage() { ... } func_i() { сделайте что-нибудь с $ 2 ... } func_m() { сделайте что-нибудь после того, как func_i завершит задачу ... } func_S() { запустите скрипт с определенной конфигурацией / разрешениями … } #Теперь начните обрабатывать позиционные параметры, пока getopts ": mSi:?" аргументы; выполните case $ args в

```
я) func_i;;
m) func_m;;
S) func_S;;
?) использование;;
```

esac выполнила смену [2)] [/ code]

Как я могу контролировать выполнение (-m) только после того, как (-i) закончит делать то, что функция объявляет для выполнения с заданным аргументом / операндом $ 2?

[1)] $ N-1
[2)] OPTIND-1

стелла, 2013/05/07 03:20 ()

Это очень помогает, большое вам спасибо!

Адриан Мику, 2013/08/23 13:41 ()

Здравствуйте,

Спасибо за отличный урок. Я обнаружил, что если последовательность while getopts ... используется внутри функции, это больше не работает, поэтому getopts необходимо использовать в "основной" части скрипта. Я не знаю, почему - может быть, у вас есть идея?

Спасибо, Адриан

палаш холкар, 08.05.12 22:20 ()

Привет,

У меня есть скрипт, похожий на команду lp: mylp filename -d имя файла для печати или mylp -d имя файла для печати

Используя getopt, я получаю имя для печати, есть ли какой-либо способ в getopt получить имя файла?

Спасибо

howto/getopts_tutorial.txt Last modified: 2018/03/21 00:07 by ffox8