

Bash-скрипты, часть 11: expect и автоматизация интерактивных утилит

<https://likegeeks.com/expect-command/>

- Блог компании RUVDS.com,
- Настройка Linux,
- Системное администрирование
- [Перевод](#)

[Bash-скрипты: начало](#)

[Bash-скрипты, часть 2: циклы](#)

[Bash-скрипты, часть 3: параметры и ключи командной строки](#)

[Bash-скрипты, часть 4: ввод и вывод](#)

[Bash-скрипты, часть 5: сигналы, фоновые задачи, управление сценариями](#)

[Bash-скрипты, часть 6: функции и разработка библиотек](#)

[Bash-скрипты, часть 7: sed и обработка текстов](#)

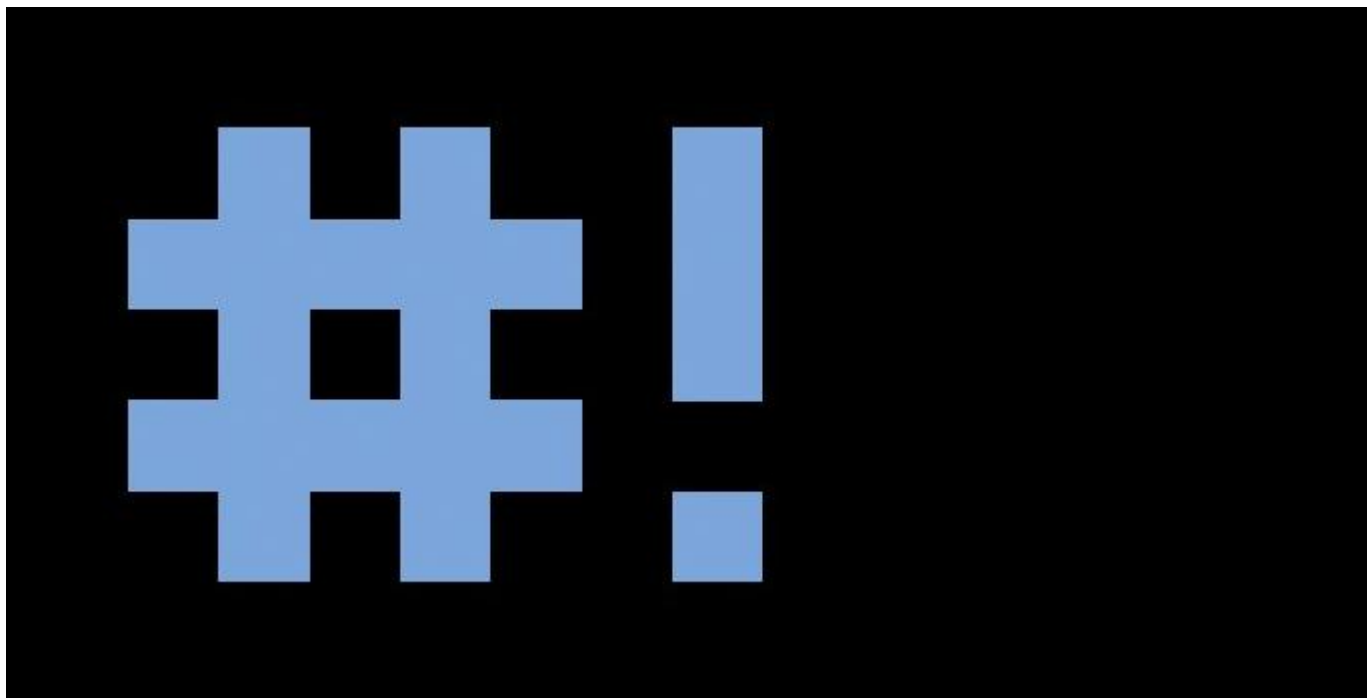
[Bash-скрипты, часть 8: язык обработки данных awk](#)

[Bash-скрипты, часть 9: регулярные выражения](#)

[Bash-скрипты, часть 10: практические примеры](#)

[Bash-скрипты, часть 11: expect и автоматизация интерактивных утилит](#)

В прошлый раз мы говорили о методике разработки bash-скриптов. Если же суммировать всё, что мы разобрали в предыдущих десяти материалах, то вы, если начинали читать их, ничего не зная о bash, теперь можете сделать уже довольно много всего полезного.



Сегодняшняя тема, заключительная в этой серии материалов, посвящена автоматизации работы с интерактивными утилитами, например, со скриптами, которые, в процессе выполнения, взаимодействуют с пользователем. В этом деле нам поможет expect — инструмент, основанный на языке Tcl.

Expect позволяет создавать программы, ожидающие вопросов от других программ и дающие им ответы. Expect можно сравнить с роботом, который способен заменить пользователя при взаимодействии со сценариями командной строки.

Habrahabr10

Промо-код для скидки в 10% на наши виртуалы

Основы expect

Если expect в вашей системе не установлен, исправить это, например, в Ubuntu, можно так:

```
$ apt-get install expect
```

В чём-то вроде CentOS установка выполняется такой командой:

```
$ yum install expect
```

Expect предоставляет набор команд, позволяющих взаимодействовать с утилитами командной строки. Вот его основные команды:

- `spawn` — запуск процесса или программы. Например, это может быть командная оболочка, [FTP](#), Telnet, ssh, scp и так далее.
- `expect` — ожидание данных, выводимых программой. При написании скрипта можно указать, какого именно вывода он ждёт и как на него нужно реагировать.
- `send` — отправка ответа. Expect-скрипт с помощью этой команды может отправлять входные данные автоматизируемой программе. Она похожа на знакомую вам команду `echo` в обычных bash-скриптах.
- `interact` — позволяет переключиться на «ручной» режим управления программой.

Автоматизация bash-скрипта

Напишем скрипт, который взаимодействует с пользователем и автоматизируем его с помощью expect. Вот код bash-скрипта `questions`:

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

```
echo "Hello, who are you?"
```

```
read $REPLY
```

```
echo "Can I ask you some questions?"
```

```
read $REPLY
```

```
echo "What is your favorite topic?"
```

```
read $REPLY
```

Теперь напомним expect-скрипт, который запустит скрипт `questions` и будет отвечать на его вопросы:

```
#!/usr/bin/expect -f
```

```
set timeout -1
```

```
spawn ./questions
```

```
expect "Hello, who are you?\r"
```

```
send -- "Im Adam\r"
```

```
expect "Can I ask you some questions?\r"
```

```
send -- "Sure\r"
```

```
expect "What is your favorite topic?\r"
```

```
send -- "Technology\r"
```

```
expect eof
```

Сохраним скрипт, дав ему имя `answerbot`.

В начале скрипта находится строка идентификации, которая, в данном случае, содержит путь к `expect`, так как интерпретировать скрипт будет именно `expect`. Во второй строке мы отключаем тайм-аут, устанавливая переменную `expect timeout` в значение `-1`. Остальной код — это и есть автоматизация работы с `bash`-скриптом.

Сначала, с помощью команды `spawn`, мы запускаем `bash`-скрипт. Естественно, тут может быть вызвана любая другая утилита командной строки. Далее задана последовательность вопросов, поступающих от `bash`-скрипта, и ответов, которые даёт на них `expect`. Получив вопрос от подпроцесса, `expect` выдаёт ему заданный ответ и ожидает следующего вопроса.

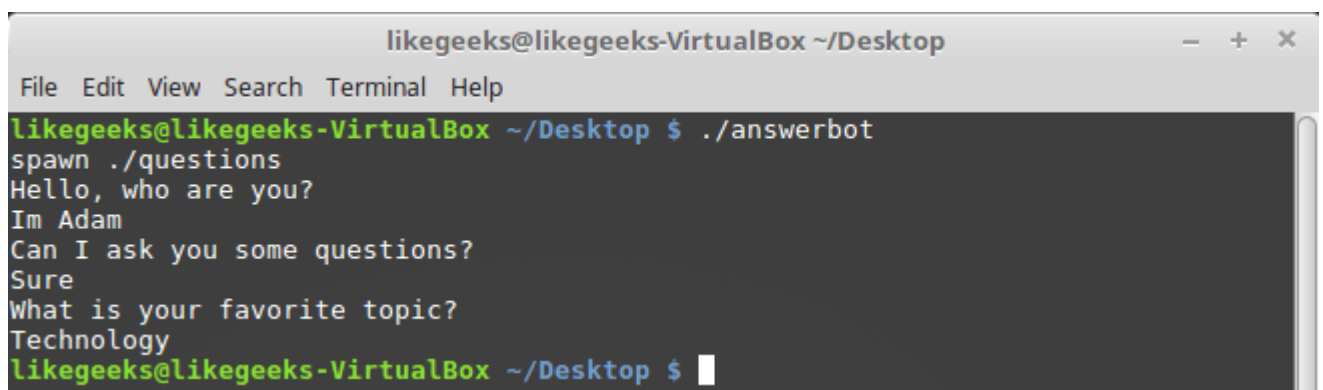
В последней команде `expect` ожидает признака конца файла, скрипт, дойдя до этой команды, завершается.

Теперь пришло время всё это опробовать. Сделаем `answerbot` исполняемым файлом:

```
$ chmod +x ./answerbot
```

И вызовем его:

```
$ ./answerbot
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./answerbot
spawn ./questions
Hello, who are you?
Im Adam
Can I ask you some questions?
Sure
What is your favorite topic?
Technology
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Expect-скрипт отвечает на вопросы bash-скрипта

Как видно, `expect`-скрипт верно ответил на вопросы `bash`-скрипта. Если на данном этапе вы столкнулись с ошибкой, вызванной тем, что неправильно указано расположение `expect`, выяснить его адрес можно так:

```
$ which expect
```

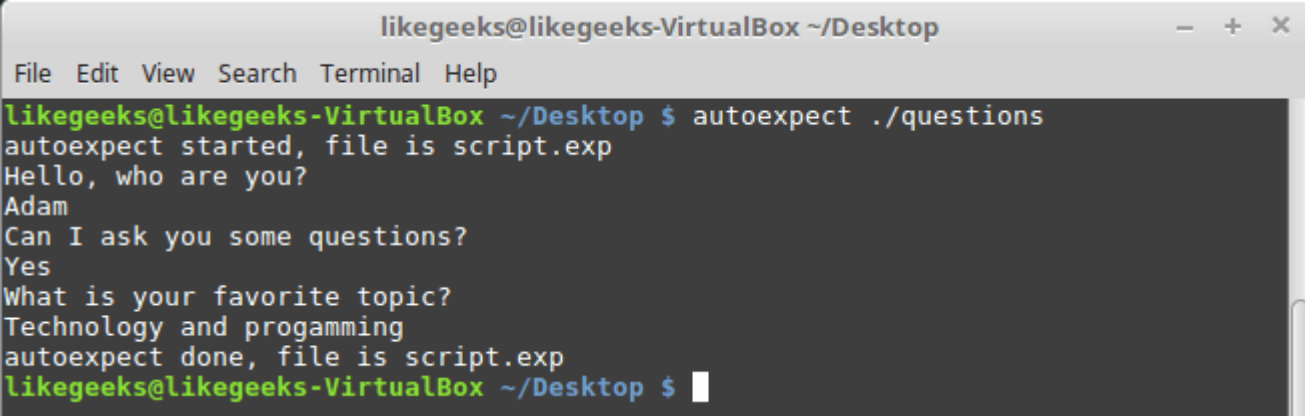
Обратите внимание на то, что после запуска скрипта `answerbot` всё происходит в полностью автоматическом режиме. То же самое можно проделать для любой утилиты командной строки. Тут надо отметить, что наш `bash`-скрипт устроен очень просто, мы точно знаем, какие именно данные он выводит, поэтому написать `expect`-скрипт для взаимодействия с ним несложно. Задача усложняется при работе с программами, которые написаны другими разработчиками. Однако, здесь на помощь приходит средство для автоматизированного создания `expect`-скриптов.

Autoexpect — автоматизированное создание `expect`-скриптов

`Autoexpect` позволяет запускать программы, которые надо автоматизировать, после чего записывает то, что они выводят, и то, что пользователь вводит, отвечая на их вопросы. Вызовем `autoexpect`, передав этой утилите имя нашего скрипта:

```
$ autoexpect ./questions
```

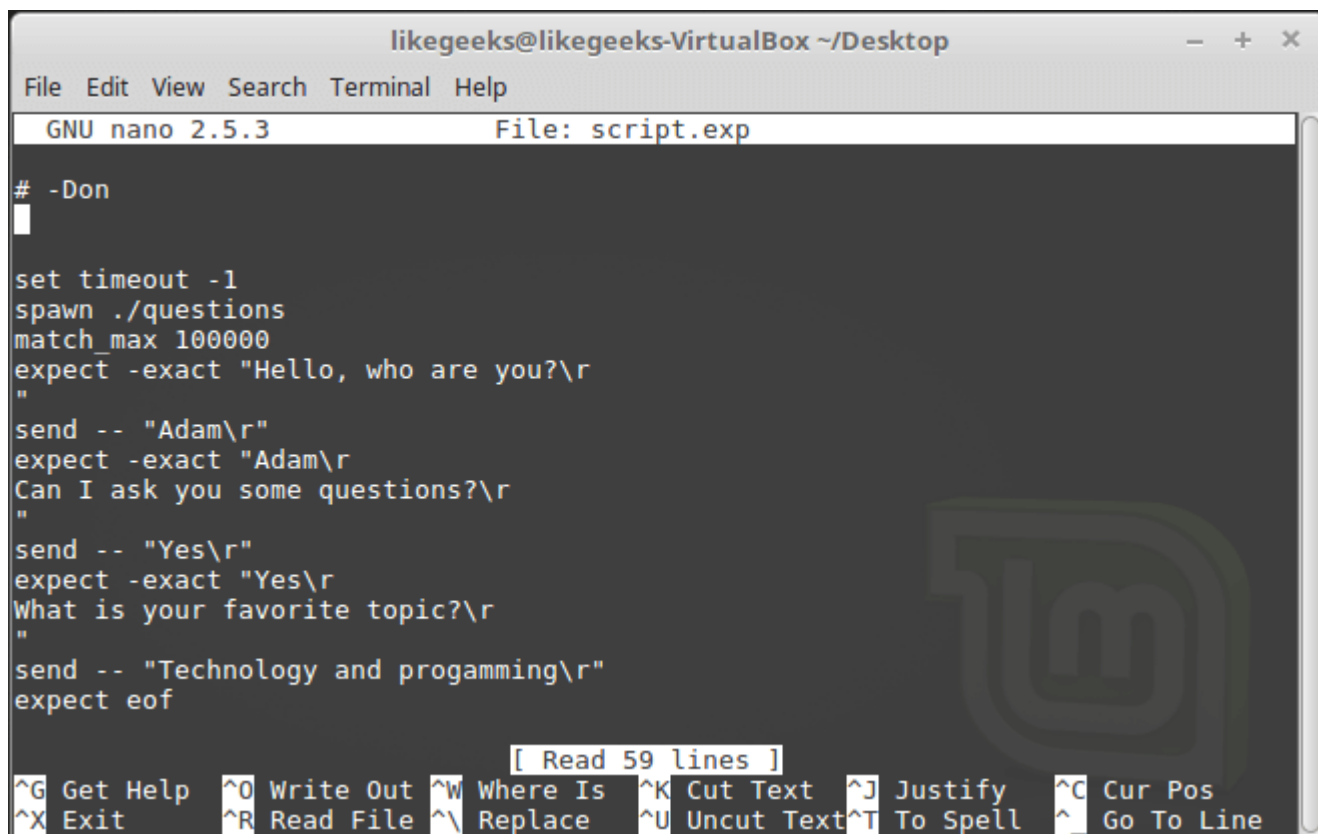
В этом режиме взаимодействие с `bash`-скриптом ничем не отличается от обычного: мы сами вводим ответы на его вопросы.



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ autoexpect ./questions
autoexpect started, file is script.exp
Hello, who are you?
Adam
Can I ask you some questions?
Yes
What is your favorite topic?
Technology and progamming
autoexpect done, file is script.exp
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Запуск `bash`-скрипта с помощью `autoexpect`

После завершения работы с `bash`-скриптом, `autoexpect` сообщит о том, что собранные данные записаны в файл `script.exp`. Взглянем на этот файл.



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
GNU nano 2.5.3 File: script.exp

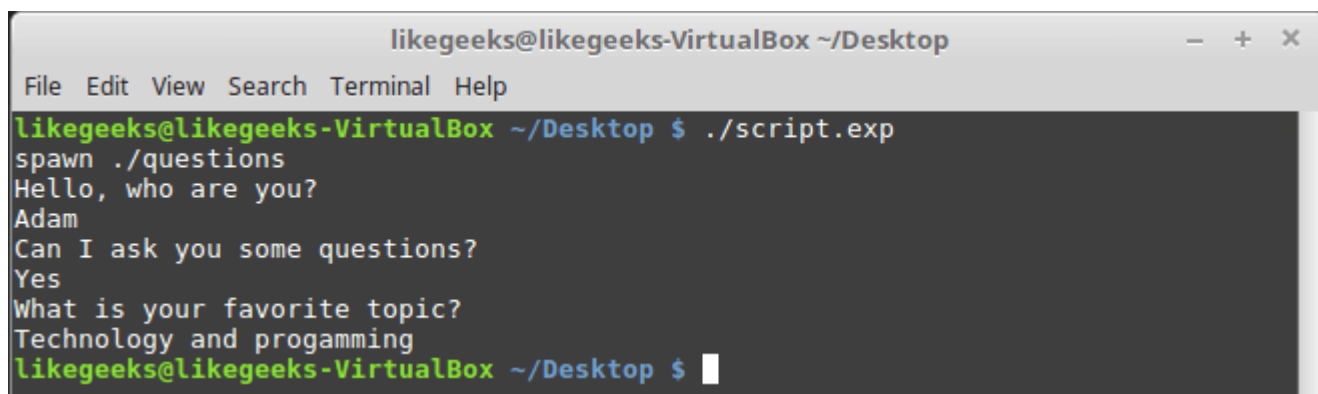
# -Don

set timeout -1
spawn ./questions
match_max 100000
expect -exact "Hello, who are you?\r"
"
send -- "Adam\r"
expect -exact "Adam\r"
Can I ask you some questions?\r
"
send -- "Yes\r"
expect -exact "Yes\r"
What is your favorite topic?\r
"
send -- "Technology and programming\r"
expect eof

[ Read 59 lines ]
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line
```

Файл script.exp

В целом, за исключением некоторых деталей, перед нами такой же скрипт, который мы писали самостоятельно. Если запустить этот скрипт, результат будет тем же.



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./script.exp
spawn ./questions
Hello, who are you?
Adam
Can I ask you some questions?
Yes
What is your favorite topic?
Technology and programming
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Запуск expect-скрипта, созданного автоматически

При записи сеансов взаимодействия с некоторыми программами, вроде FTP-клиентов, вы можете столкнуться с тем, что они используют в выводимых данных сведения о времени проведения операции, или выводят данные, отражающие процесс выполнения неких продолжительных действий. В целом, речь идёт о том, что вывод программы при каждом её запуске, правильно воспринимаемый человеком и вызывающий ввод одних и тех же ответов, будет, в тех же условиях, выглядеть по-новому для expect.

Если в expect-скрипте строки, ожидаемые от такой программы, будут жёстко

зафиксированы, такой скрипт не сможет нормально работать. Справиться с этим можно, либо удалив из ехрест-скрипта данные, которые выглядят по-новому при каждом запуске программы, либо используя шаблоны, пользуясь которыми, ехрест сможет правильно понять то, что хочет от него программа.

Как видите, `autoexhrest` — это весьма полезный инструмент, но и он не лишён недостатков, исправить которые можно только вручную. Поэтому продолжим осваивать язык ехрест-скриптов.

Работа с переменными и параметрами командной строки

Для объявления переменных в ехрест-скриптах используется команда `set`. Например, для того, чтобы присвоить значение 5 переменной `VAR1`, используется следующая конструкция:

```
set VAR1 5
```

Для доступа к значению переменной перед её именем надо добавить знак доллара — `$`. В нашем случае это будет выглядеть как `$VAR1`.

Для того, чтобы получить доступ к аргументам командной строки, с которыми вызван ехрест-скрипт, можно поступить так:

```
set VAR [lindex $argv 0]
```

Тут мы объявляем переменную `VAR` и записываем в неё указатель на первый аргумент командной строки, `$argv 0`.

Для целей обновлённого ехрест-скрипта мы собираемся записать значение первого аргумента, представляющее собой имя пользователя, которое будет использовано в программе, в переменную `my_name`. Второй аргумент, символизирующий то, что пользователю нравится, попадёт в переменную `my_favorite`. В результате объявление переменных будет выглядеть так:

```
set my_name [lindex $argv 0]
```

```
set my_favorite [lindex $argv 1]
```

Отредактируем скрипт `answerbot`, приведя его к такому виду:

```
#!/usr/bin/expect -f
```

```
set my_name [lindex $argv 0]
```

```
set my_favorite [lindex $argv 1]
```

```
set timeout -1
```

```
spawn ./questions
```

```
expect "Hello, who are you?\r"
```

```
send -- "Im $my_name\r"
```

```
expect "Can I ask you some questions?\r"
```

```
send -- "Sure\r"
```

```
expect "What is your favorite topic?\r"
```

```
send -- "$my_favorite\r"
```

```
expect eof
```

Запустим его, передав в качестве первого параметра `SomeName`, в качестве второго — `Programming`:

```
$ ./answerbot SomeName Programming
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./answerbot SomeName Programming
spawn ./questions
Hello, who are you?
Im SomeName
Can I ask you some questions?
Sure
What is your favorite topic?
Programming
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Ехпекст-скрипт, использующий переменные и параметры командной строки

Как видите, всё работает так, как ожидалось. Теперь ехпекст-скрипт отвечает на вопросы bash-скрипта, пользуясь переданными ему параметрами командной строки.

Ответы на разные вопросы, которые могут появиться в одном и том же месте

Если автоматизируемая программа может, в одной ситуации, выдать одну строку, а в другой, в том же самом месте — другую, в ехпекст можно использовать блоки, заключённые в фигурные скобки и содержащие варианты реакции скрипта на разные данные, полученные от программы. Выглядит это так:

```
expect {

    "something" { send -- "send this\r" }

    "*another" { send -- "send another\r" }

}
```

Здесь, если ехпекст-скрипт увидит строку «something», он отправит ответ «send this». Если же это будет некая строка, оканчивающаяся на «another», он отправит ответ «send another».

Напишем новый скрипт, записав его в файл `questions`, случайным образом задающий в одном и том же месте разные вопросы:

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

```
let number=$RANDOM
```

```
if [ $number -gt 25000 ]
```

```
then
```

```
echo "What is your favorite topic?"
```

```
else
```

```
echo "What is your favorite movie?"
```

```
fi
```

```
read $REPLY
```

Тут мы генерируем случайное число при каждом запуске скрипта, и, проанализировав его, выводим один из двух вопросов.

Для автоматизации такого скрипта нам и пригодится вышеописанная конструкция:

```
#!/usr/bin/expect -f
```

```
set timeout -1
```

```
spawn ./questions
```

```
expect {
```

```
    "*topic?" { send -- "Programming\r" }
```

```
    "*movie?" { send -- "Star wars\r" }
```

```
}
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./answerbot
spawn ./questions
What is your favorite movie?
Star wars
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./answerbot
spawn ./questions
What is your favorite topic?
Programming
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Ответы на разные вопросы, появляющиеся в одном и том же месте

Как видно, когда автоматизированный скрипт выводит строку, оканчивающуюся на «topic?», expect-скрипт передаёт ему строку «Programming». Получив в том же месте, при другом запуске программы, вопрос, оканчивающийся на «movie?», expect-скрипт отвечает: «Star wars». Это очень полезная техника.

Условный оператор

Expect поддерживает условный оператор `if-else` и другие управляющие конструкции. Вот пример использования условного оператора:

```
#!/usr/bin/expect -f
```

```
set TOTAL 1
```

```
if { $TOTAL < 5 } {
```

```
puts "\nTOTAL is less than 5\n"
```

```
} elseif { $TOTAL > 5 } {
```

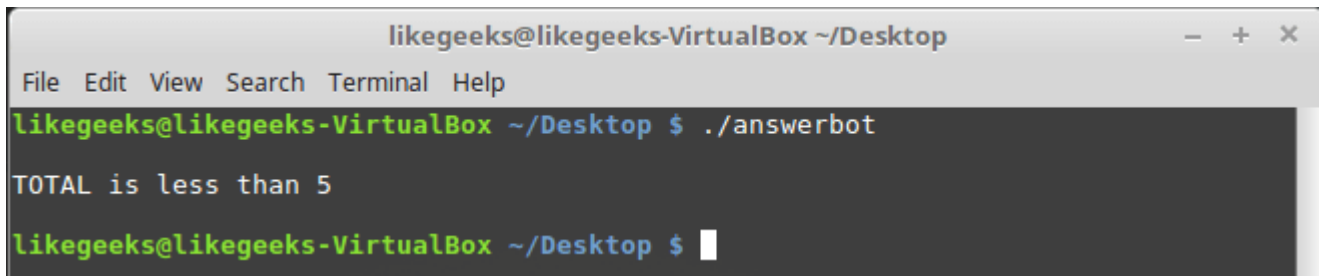
```
puts "\nTOTAL greater than 5\n"
```

```
} else {
```

```
puts "\nTOTAL is equal to 5\n"
```

```
}
```

```
expect eof
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./answerbot
TOTAL is less than 5
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Условный оператор в expect

Тут мы присваиваем переменной `TOTAL` некое число, после чего проверяем его и выводим текст, зависящий от результата проверки.

Обратите внимание на конфигурацию фигурных скобок. Очередная открывающая скобка должна быть расположена на той же строке, что и предыдущие конструкции.

Цикл while

Циклы `while` в `expect` очень похожи на те, что используются в обычных `bash`-скриптах, но, опять же, тут применяются фигурные скобки:

```
#!/usr/bin/expect -f
```

```
set COUNT 0
```

```
while { $COUNT <= 5 } {
```

```
puts "\nCOUNT is currently at $COUNT"
```

```
set COUNT [ expr $COUNT + 1 ]
```

```
}
```

```
puts ""
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./answerbot
COUNT is currently at 0
COUNT is currently at 1
COUNT is currently at 2
COUNT is currently at 3
COUNT is currently at 4
COUNT is currently at 5
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Цикл while в expect

Цикл for

Цикл `for` в `expect` устроен по-особому. В начале цикла, в самостоятельных парах фигурных скобок, надо указать переменную-счётчик, условие прекращения цикла и правило модификации счётчика. Затем, опять же в фигурных скобках, идёт тело цикла:

```
#!/usr/bin/expect -f
```

```
for {set COUNT 0} {$COUNT <= 5} {incr COUNT} {
```

```
puts "\nCOUNT is at $COUNT"
```

```
}
```

```
puts ""
```

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./answerbot

COUNT is at 0
COUNT is at 1
COUNT is at 2
COUNT is at 3
COUNT is at 4
COUNT is at 5
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Цикл for в expect

Объявление и использование функций

Ехпекст позволяет программисту объявлять функции, используя ключевое слово `proc`:

```
proc myfunc { MY_COUNT } {  
    set MY_COUNT [expr $MY_COUNT + 1]  
    return "$MY_COUNT"  
}
```

Вот как выглядит ехпекст-скрипт, в котором используется объявленная в нём же функция:

```
#!/usr/bin/expect -f  
  
proc myfunc { MY_COUNT } {  
    set MY_COUNT [expr $MY_COUNT + 1]
```

```
return "$MY_COUNT"
```

```
}
```

```
set COUNT 0
```

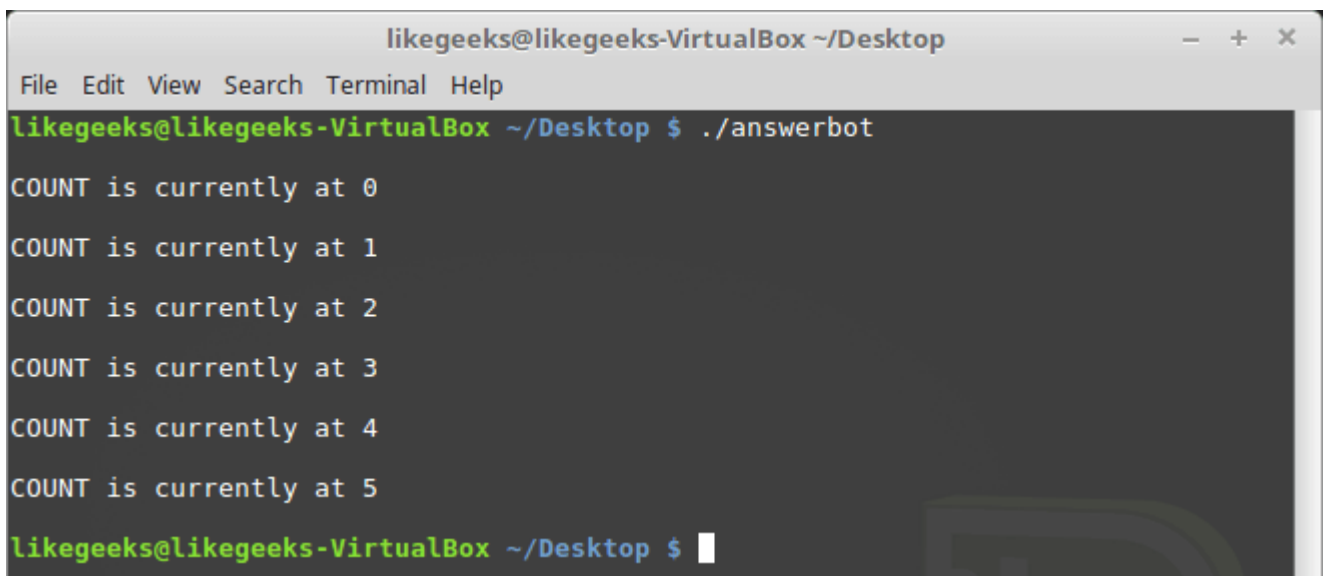
```
while {$COUNT <= 5} {
```

```
puts "\nCOUNT is currently at $COUNT"
```

```
set COUNT [myfunc $COUNT]
```

```
}
```

```
puts ""
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./answerbot
COUNT is currently at 0
COUNT is currently at 1
COUNT is currently at 2
COUNT is currently at 3
COUNT is currently at 4
COUNT is currently at 5
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Функции в exspect

Команда interact

Случается так, что автоматизируемые с помощью exspect программы требуют ввода конфиденциальных данных, вроде паролей, которые вам не хотелось бы хранить в виде обычного текста в коде скрипта. В подобной ситуации можно воспользоваться командой `interact`, которая позволит вам, автоматизировав некую часть взаимодействия с программой, самостоятельно ввести, скажем,

пароль, а потом опять передать управление expect.

Когда выполняется эта команда, expect-скрипт переключается на чтение ответа на вопрос программы с клавиатуры, вместо того, чтобы передавать ей ранее записанные в нём данные.

Вот bash-скрипт, в общем-то, точно такой же, как мы рассматривали ранее, но теперь ожидающий ввод пароля в ответ на один из своих вопросов:

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

```
echo "Hello, who are you?"
```

```
read $REPLY
```

```
echo "What is your password?"
```

```
read $REPLY
```

```
echo "What is your favorite topic?"
```

```
read $REPLY
```

Напишем expect-скрипт, который, когда ему предлагают предоставить пароль, передаёт управление нам:

```
#!/usr/bin/expect -f
```

```
set timeout -1
```

```
spawn ./questions
```

```
expect "Hello, who are you?\r"
```

```
send -- "Hi Im Adam\r"
```

```
expect "*password?\r"
```



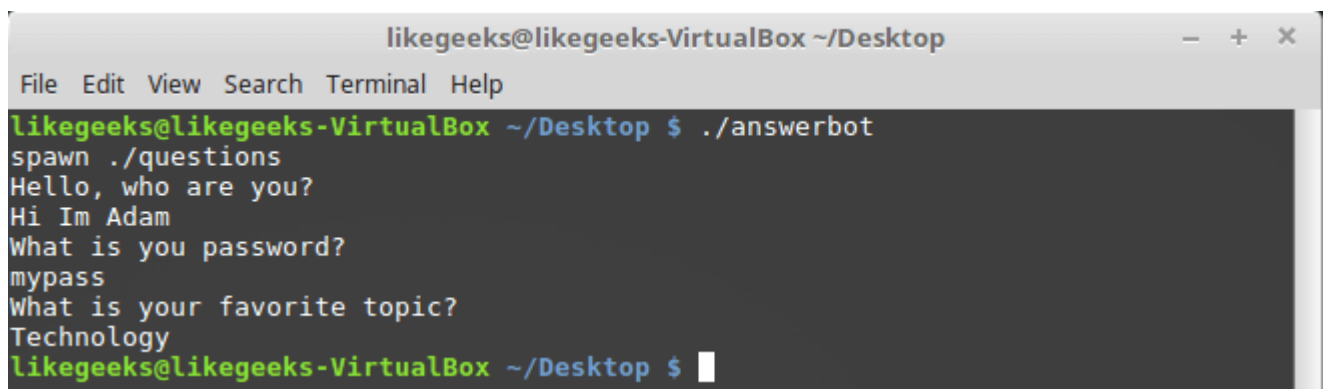
```
interact ++ return
```

```
send "\r"
```

```
expect "*topic?\r"
```

```
send -- "Technology\r"
```

```
expect eof
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./answerbot
spawn ./questions
Hello, who are you?
Hi Im Adam
What is you password?
mypass
What is your favorite topic?
Technology
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Команда *interact* в expect-скрипте

Встретив команду `interact`, expect-скрипт остановится, предоставив нам возможность ввести пароль. После ввода пароля надо ввести «++» и expect-скрипт продолжит работу, снова получив управление.

Итоги

Возможностями expect можно пользоваться в программах, написанных на разных языках программирования благодаря соответствующим библиотекам. Среди этих языков — C#, Java, Perl, Python, Ruby, и другие. То, что expect доступен для разных сред разработки — далеко не случайность. Всё дело в том, что это действительно важный и полезный инструмент, который используют для решения множества задач. Здесь и проверка качества ПО, и выполнение различных работ по сетевому администрированию, автоматизация передачи файлов, автоматическая установка обновлений и многое другое.

Освоив этот материал, вы ознакомились с основными концепциями expect и научились пользоваться инструментом autoexpect для автоматического формирования скриптов. Теперь вы вполне можете продолжить изучение expect, воспользовавшись дополнительными источниками. Вот — [сборник](#) учебных и справочных материалов. Вот — достойная внимания серия из трёх статей ([1](#), [2](#), [3](#)). А вот — [официальная страница](#) expect, на которой можно найти ссылки на исходный код программы и список публикаций.

На этом мы завершаем серию материалов о bash-скриптах. Надеемся, её одиннадцать частей, а также бессчётное число комментариев к ним, помогли в достижении цели тем, кто хотел научиться писать сценарии командной строки.