

Лабораторная работа 5 #4

Bash-скрипты, часть 4: ввод и вывод

В прошлый раз, в третьей части лабораторной работы по bash-скриптам, говорилось о параметрах командной строки и ключах. Следующий этап этой темы — ввод, вывод, и всё, что с этим связано.

Вы уже знакомы с двумя методами работы с тем, что выводят сценарии командной строки:

- Отображение выводимых данных на экране.
- Перенаправление вывода в файл.

Иногда что-то надо показать на экране, а что-то — записать в файл, поэтому нужно разобраться с тем, как в Linux обрабатывается ввод и вывод, а значит — научиться отправлять результаты работы сценариев туда, куда нужно. Начнём с разговора о стандартных дескрипторах файлов.

Стандартные дескрипторы файлов

Всё в Linux — это файлы, в том числе — ввод и вывод. Операционная система идентифицирует файлы с использованием дескрипторов.

Каждому процессу позволено иметь до девяти открытых дескрипторов файлов. Оболочка bash резервирует первые три дескриптора с идентификаторами 0, 1 и 2. Вот что они означают:

- 0, STDIN — стандартный поток ввода.
- 1, STDOUT — стандартный поток вывода.
- 2, STDERR — стандартный поток ошибок.

Эти три специальных дескриптора обрабатывают ввод и вывод данных в сценарии.

Вам нужно как следует разобраться в стандартных потоках. Их можно сравнить с фундаментом, на котором строится взаимодействие скриптов с внешним миром. Рассмотрим подробности о них.

STDIN

STDIN — это стандартный поток ввода оболочки. Для терминала стандартный ввод — это клавиатура. Когда в сценариях используют символ перенаправления ввода — `<`, Linux заменяет дескриптор файла стандартного ввода на тот, который указан в команде. Система читает файл и обрабатывает данные так, будто они введены с клавиатуры.

Многие команды `bash` принимают ввод из STDIN, если в командной строке не указан файл, из которого надо брать данные. Например, это справедливо для команды `cat`.

Когда вы вводите команду `cat` в командной строке, не задавая параметров, она принимает ввод из STDIN. После того, как вы вводите очередную строку, `cat` просто выводит её на экран.

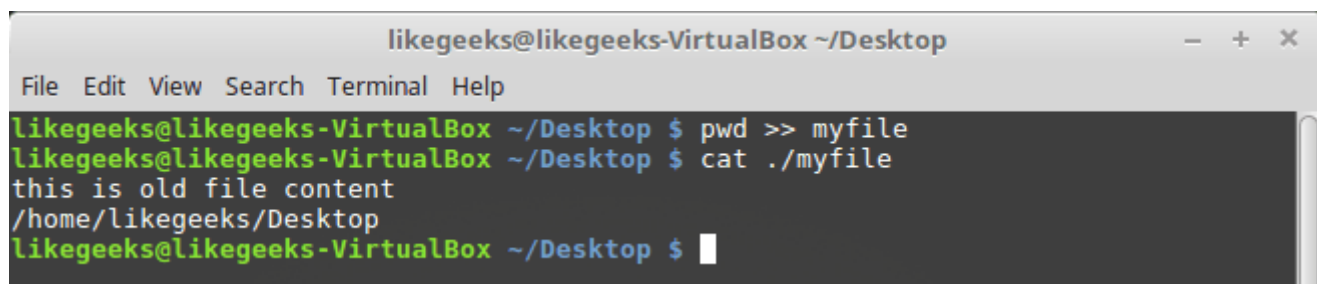
STDOUT

STDOUT — стандартный поток вывода оболочки. По умолчанию это — экран. Большинство `bash`команд выводят данные в STDOUT, что приводит к их появлению в консоли. Данные можно перенаправить в файл, присоединяя их к его содержимому, для этого служит команда `>>`.

Итак, у нас есть некий файл с данными, к которому мы можем добавить другие данные с помощью этой команды:

```
pwd >> myfile
```

То, что выведет `pwd`, будет добавлено к файлу `myfile`, при этом уже имеющиеся в нём данные никуда не денутся.



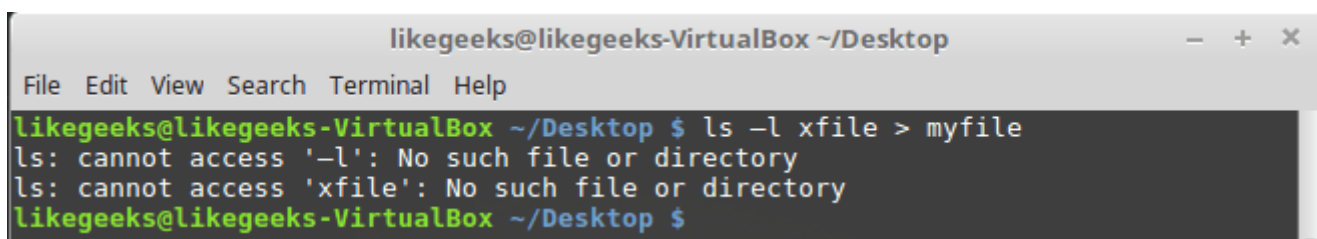
```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ pwd >> myfile
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./myfile
this is old file content
/home/likegeeks/Desktop
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Перенаправление вывода команды в файл

Пока всё хорошо, но что если попытаться выполнить что-то вроде показанного ниже, обратившись к несуществующему файлу `xfile`, задумывая всё это для того, чтобы в файл `myfile` попало сообщение об ошибке.

```
ls -l xfile > myfile
```

После выполнения этой команды мы увидим сообщения об ошибках на экране.



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ls -l xfile > myfile
ls: cannot access '-l': No such file or directory
ls: cannot access 'xfile': No such file or directory
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Попытка обращения к несуществующему файлу

При попытке обращения к несуществующему файлу генерируется ошибка, но оболочка не перенаправила сообщения об ошибках в файл, выведя их на экран. Но мы-то хотели, чтобы сообщения об ошибках попали в файл. Что делать? Ответ прост — воспользоваться третьим стандартным дескриптором.

STDERR

STDERR представляет собой стандартный поток ошибок оболочки. По умолчанию этот дескриптор указывает на то же самое, на что указывает STDOUT, именно поэтому при возникновении ошибки мы видим сообщение на экране.

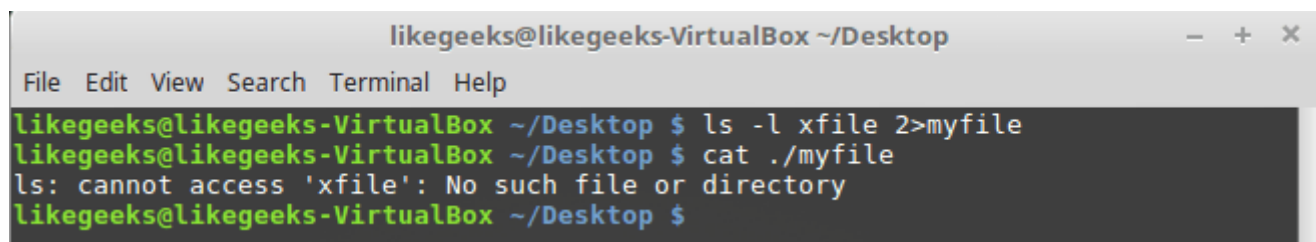
Итак, предположим, что надо перенаправить сообщения об ошибках, скажем, в лог-файл, или куда-нибудь ещё, вместо того, чтобы выводить их на экран.

Перенаправление потока ошибок

Как вы уже знаете, дескриптор файла STDERR — 2. Мы можем перенаправить ошибки, разместив этот дескриптор перед командой перенаправления:

```
ls -l xfile 2>myfile  
cat ./myfile
```

Сообщение об ошибке теперь попадёт в файл myfile.

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The terminal shows the following commands and output:

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ls -l xfile 2>myfile  
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./myfile  
ls: cannot access 'xfile': No such file or directory  
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

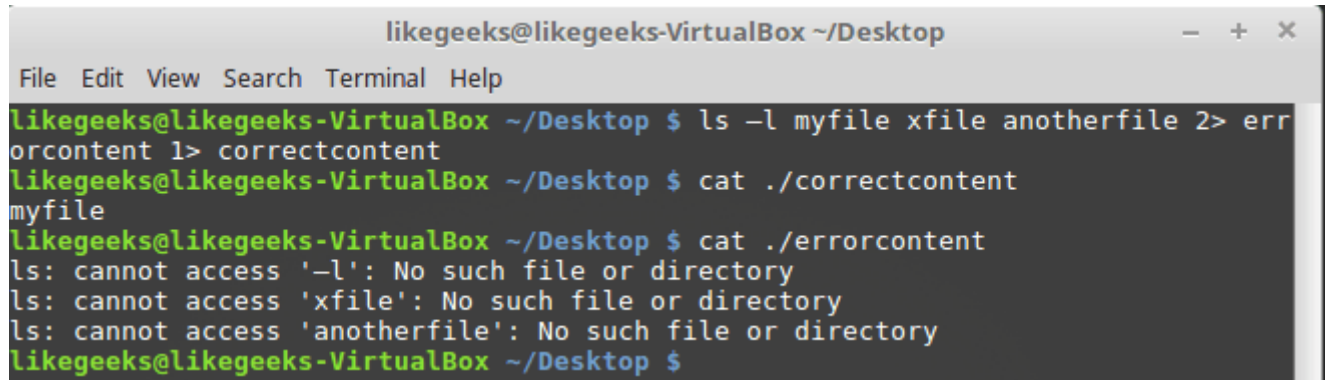
Перенаправление сообщения об ошибке в файл

Перенаправление потоков ошибок и вывода

При написании сценариев командной строки может возникнуть ситуация, когда нужно организовать и перенаправление сообщений об ошибках, и перенаправление стандартного вывода. Для того, чтобы этого добиться, нужно использовать команды перенаправления для соответствующих дескрипто-

ров с указанием файлов, куда должны попадать ошибки и стандартный вывод:

```
ls -l myfile xfile anotherfile 2> errorcontent 1> correctcontent
```

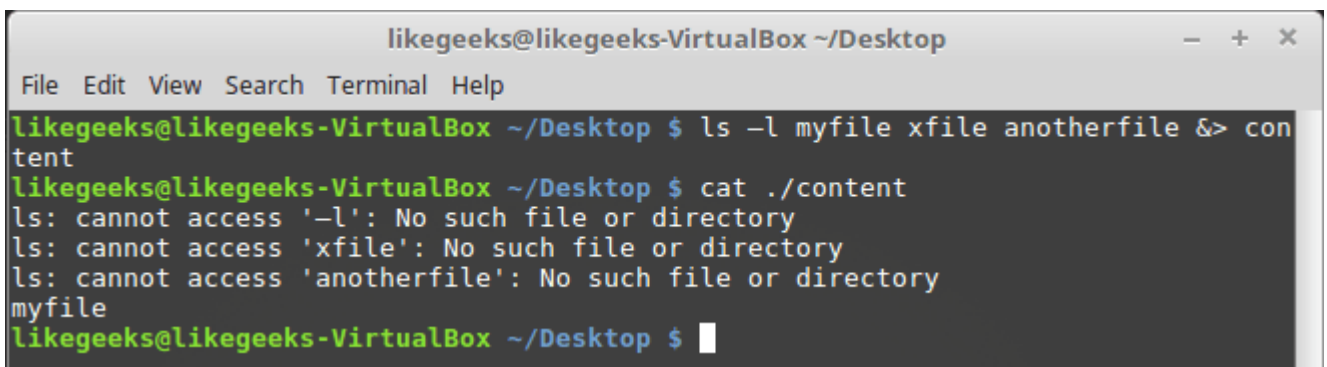
A terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ls -l myfile xfile anotherfile 2> errorcontent 1> correctcontent
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./correctcontent
myfile
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./errorcontent
ls: cannot access '-l': No such file or directory
ls: cannot access 'xfile': No such file or directory
ls: cannot access 'anotherfile': No such file or directory
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Перенаправление ошибок и стандартного вывода

Оболочка перенаправит то, что команда `ls` обычно отправляет в `STDOUT`, в файл `correctcontent` благодаря конструкции `1>`. Сообщения об ошибках, которые попали бы в `STDERR`, оказываются в файле `errorcontent` из-за команды перенаправления `2>`.

Если надо, и `STDERR`, и `STDOUT` можно перенаправить в один и тот же файл, воспользовавшись командой `&>`:

A terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the following commands and output:

```
likegeeks@likegeeks-VirtualBox ~/Desktop $ ls -l myfile xfile anotherfile &> content
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./content
ls: cannot access '-l': No such file or directory
ls: cannot access 'xfile': No such file or directory
ls: cannot access 'anotherfile': No such file or directory
myfile
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Перенаправление `STDERR` и `STDOUT` в один и тот же файл

После выполнения команды то, что предназначено для `STDERR` и `STDOUT`, оказывается в файле `content`.

Перенаправление вывода в скриптах

Существует два метода перенаправления вывода в сценариях командной строки:

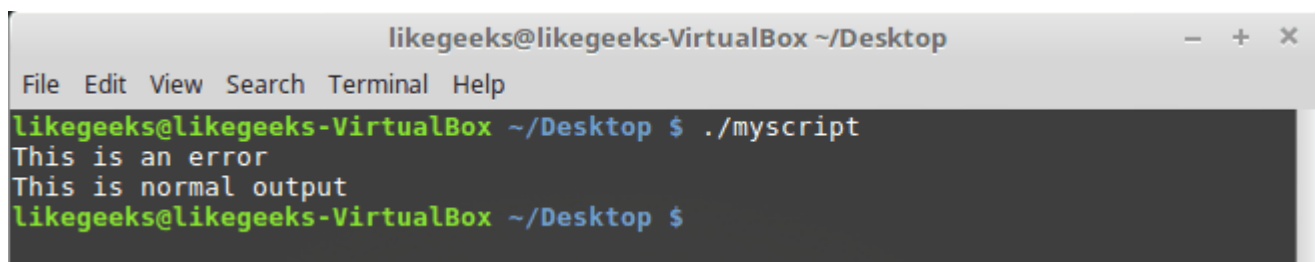
- Временное перенаправление, или перенаправление вывода одной строки.
- Постоянное перенаправление, или перенаправление всего вывода в скрипте либо в какой-то его части.

Временное перенаправление вывода

В скрипте можно перенаправить вывод отдельной строки в STDERR. Для того, чтобы это сделать, достаточно использовать команду перенаправления, указав дескриптор STDERR, при этом перед номером дескриптора надо поставить символ амперсанда (&):

```
#!/bin/bash
echo "This is an error" >&2
echo "This is normal output"
```

Если запустить скрипт, обе строки попадут на экран, так как, как вы уже знаете, по умолчанию ошибки выводятся туда же, куда и обычные данные.



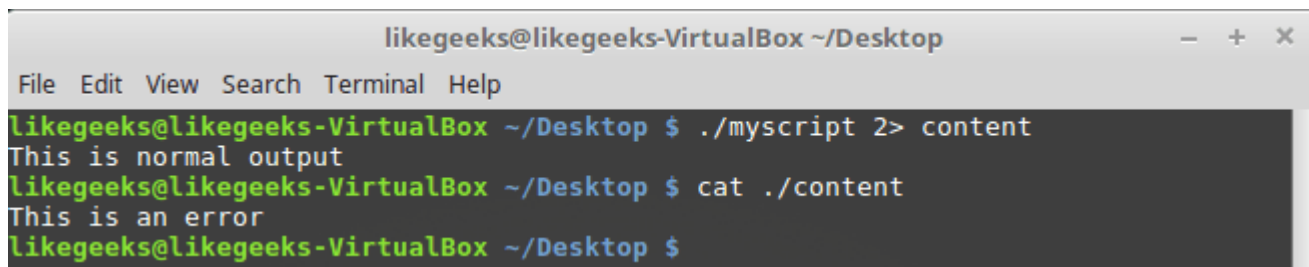
```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
This is an error
This is normal output
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Временное перенаправление

Запустим скрипт так, чтобы вывод STDERR попадал в файл.

```
./myscript 2> myfile
```

Как видно, теперь обычный вывод делается в консоль, а сообщения об ошибках попадают в файл.

A terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. It shows the execution of a script './myscript' with output redirection '2> content'. The script prints 'This is normal output' to the console and 'This is an error' to the file 'content'.

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript 2> content
This is normal output
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./content
This is an error
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

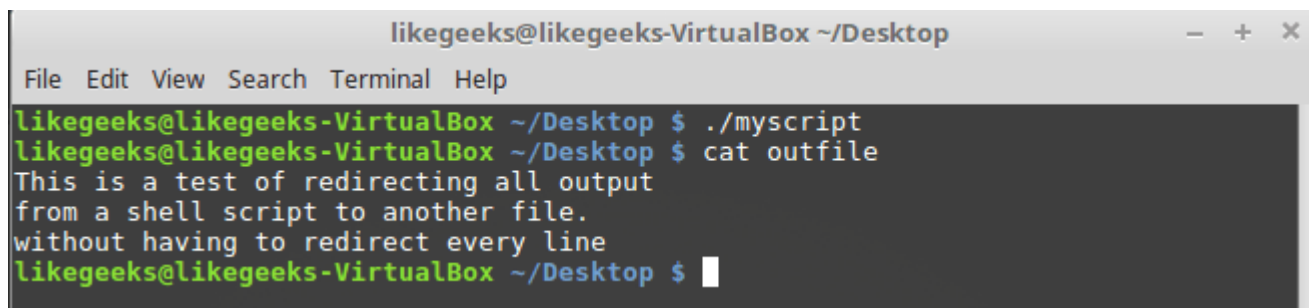
Сообщения об ошибках записываются в файл

Постоянное перенаправление вывода

Если в скрипте нужно перенаправлять много выводимых на экран данных, добавлять соответствующую команду к каждому вызову `echo` неудобно. Вместо этого можно задать перенаправление вывода в определённый дескриптор на время выполнения скрипта, воспользовавшись командой `exec`:

```
#!/bin/bash
exec 1>outfile
echo "This is a test of redirecting all output"
echo "from a shell script to another file."
echo "without having to redirect every line"
```

Запустим скрипт.

A terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. It shows the execution of a script './myscript' which has redirected all output to 'outfile'. The script prints three lines of text, which are then shown in the file 'outfile' using 'cat outfile'.

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat outfile
This is a test of redirecting all output
from a shell script to another file.
without having to redirect every line
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

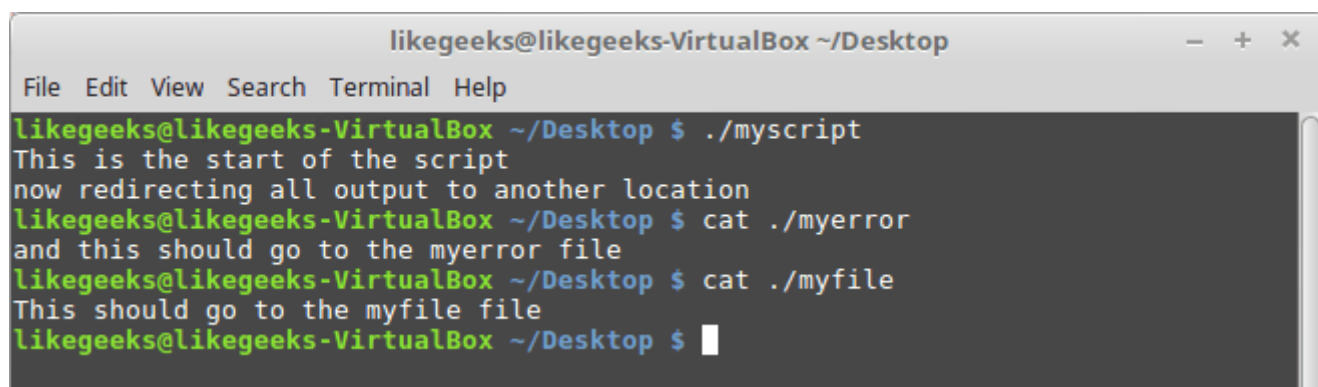
Перенаправление всего вывода в файл

Если посмотреть файл, указанный в команде перенаправления вывода, окажется, что всё, что выводилось командами `echo`, попало в этот файл.

Команду `exec` можно использовать не только в начале скрипта, но и в других местах:

```
#!/bin/bash
exec 2>myerror
echo "This is the start of the script"
echo "now redirecting all output to another location"
exec 1>myfile
echo "This should go to the myfile file"
echo "and this should go to the myerror file" >&2
```

Вот что получится после запуска скрипта и просмотра файлов, в которые мы перенаправляли вывод.

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The terminal shows the execution of a script './myscript' which outputs 'This is the start of the script' and 'now redirecting all output to another location'. Then, the user runs 'cat ./myerror' and 'cat ./myfile' to show the redirected output. The output of 'cat ./myerror' is 'and this should go to the myerror file' and the output of 'cat ./myfile' is 'This should go to the myfile file'. The prompt is currently at 'likegeeks@likegeeks-VirtualBox ~/Desktop \$' with a cursor.

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
This is the start of the script
now redirecting all output to another location
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./myerror
and this should go to the myerror file
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./myfile
This should go to the myfile file
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Перенаправление вывода в разные файлы

Сначала команда `exec` задаёт перенаправление вывода из `STDERR` в файл `myerror`. Затем вывод нескольких команд `echo` отправляется в `STDOUT` и выводится на экран. После этого команда `exec` задаёт отправку того, что попадает в `STDOUT`, в файл `myfile`, и, наконец, мы пользуемся командой перенаправления в `STDERR` в команде `echo`, что приводит к записи соответствующей строки в файл `myerror`.

Освоив это, вы сможете перенаправлять вывод туда, куда нужно. Теперь поговорим о перенаправлении ввода.

Перенаправление ввода в скриптах

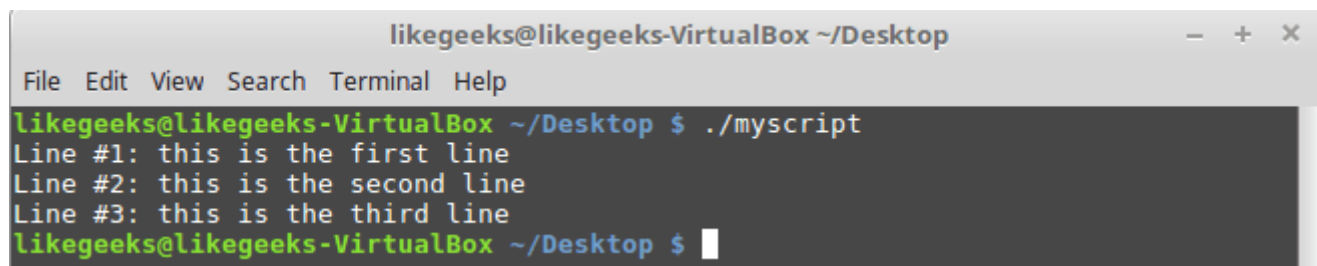
Для перенаправления ввода можно воспользоваться той же методикой, которую мы применяли для перенаправления вывода. Например, команда `exec` позволяет сделать источником данных для STDIN какой-нибудь файл:

```
exec 0< myfile
```

Эта команда указывает оболочке на то, что источником вводимых данных должен стать файл `myfile`, а не обычный STDIN. Посмотрим на перенаправление ввода в действии:

```
#!/bin/bash
exec 0< testfile
count=1
while read line
do
echo "Line #${count}: $line"
count=$(( $count + 1 ))
done
```

Вот что появится на экране после запуска скрипта.

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The terminal shows the execution of a script named 'myscript'. The script reads from a file named 'testfile' (via 'exec 0< testfile') and prints three lines: 'Line #1: this is the first line', 'Line #2: this is the second line', and 'Line #3: this is the third line'. The prompt shows the user has finished running the script and is at the shell prompt again.

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
Line #1: this is the first line
Line #2: this is the second line
Line #3: this is the third line
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Перенаправление ввода

В одном из предыдущих материалов вы узнали о том, как использовать команду `read` для чтения данных, вводимых пользователем с клавиатуры. Если перенаправить ввод, сделав источником данных файл, то команда `read`, при попытке прочитать данные из `STDIN`, будет читать их из файла, а не с клавиатуры.

Некоторые администраторы Linux используют этот подход для чтения и последующей обработки лог-файлов.

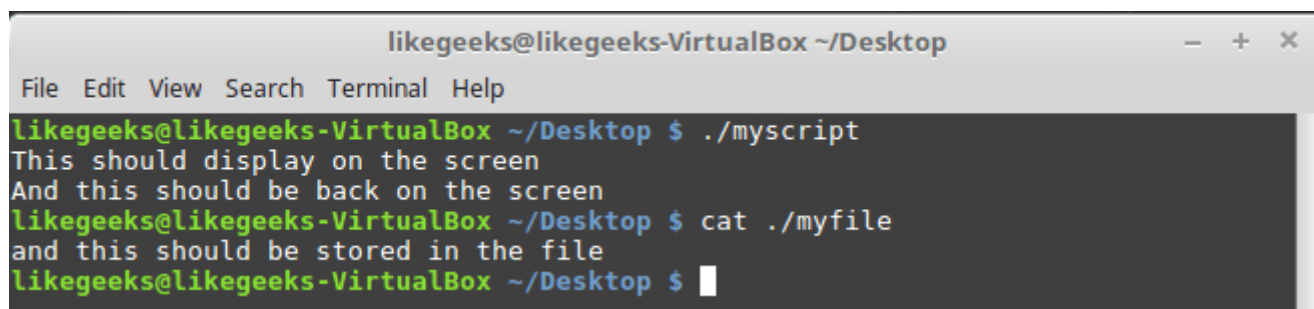
Создание собственного перенаправления вывода

Перенаправляя ввод и вывод в сценариях, вы не ограничены тремя стандартными дескрипторами файлов. Как уже говорилось, можно иметь до девяти открытых дескрипторов. Остальные шесть, с номерами от 3 до 8, можно использовать для перенаправления ввода или вывода. Любой из них можно назначить файлу и использовать в коде скрипта.

Назначить дескриптор для вывода данных можно, используя команду `exec`:

```
#!/bin/bash
exec 3>myfile
echo "This should display on the screen"
echo "and this should be stored in the file" >&3
echo "And this should be back on the screen"
```

После запуска скрипта часть вывода попадёт на экран, часть — в файл с дескриптором 3.

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The terminal shows the execution of a script './myscript' which outputs 'This should display on the screen' and 'And this should be back on the screen'. Then, the command 'cat ./myfile' is executed, showing the output 'and this should be stored in the file'. The prompt is currently at 'likegeeks@likegeeks-VirtualBox ~/Desktop \$' with a cursor.

```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
This should display on the screen
And this should be back on the screen
likegeeks@likegeeks-VirtualBox ~/Desktop $ cat ./myfile
and this should be stored in the file
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Перенаправление вывода, используя собственный дескриптор

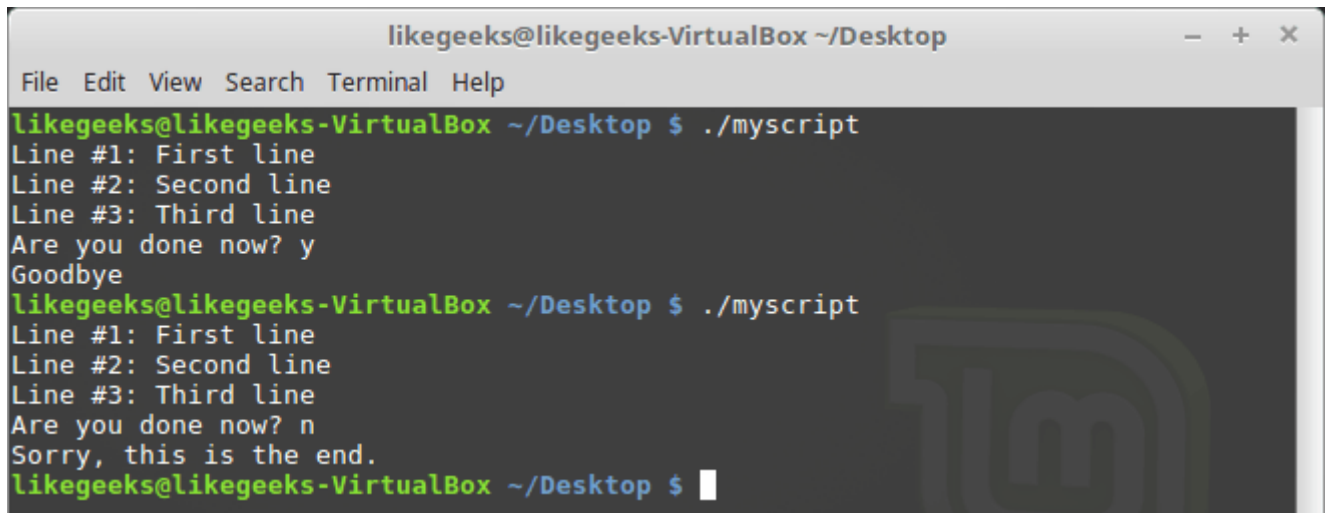
Создание дескрипторов файлов для ввода данных

Перенаправить ввод в скрипте можно точно так же, как и вывод. Сохраните STDIN в другом дескрипторе, прежде чем перенаправлять ввод данных.

После окончания чтения файла можно восстановить STDIN и пользоваться им как обычно:

```
#!/bin/bash
exec 6<&0
exec 0< myfile
count=1
while read line
do
echo "Line #$count: $line"
count=$(( $count + 1 ))
done
exec 0<&6
read -p "Are you done now? " answer
case $answer in
y) echo "Goodbye";;
n) echo "Sorry, this is the end.";;
esac
```

Испытаем сценарий.



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
Line #1: First line
Line #2: Second line
Line #3: Third line
Are you done now? y
Goodbye
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
Line #1: First line
Line #2: Second line
Line #3: Third line
Are you done now? n
Sorry, this is the end.
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Перенаправление ввода

В этом примере дескриптор файла 6 использовался для хранения ссылки на STDIN. Затем было сделано перенаправление ввода, источником данных для STDIN стал файл. После этого входные данные для команды `read` поступали из перенаправленного STDIN, то есть из файла.

После чтения файла мы возвращаем STDIN в исходное состояние, перенаправляя его в дескриптор 6. Теперь, для того, чтобы проверить, что всё работает правильно, скрипт задаёт пользователю вопрос, ожидает ввода с клавиатуры и обрабатывает то, что введено.

Заккрытие дескрипторов файлов

Оболочка автоматически закрывает дескрипторы файлов после завершения работы скрипта. Однако, в некоторых случаях нужно закрывать дескрипторы вручную, до того, как скрипт закончит работу. Для того, чтобы закрыть дескриптор, его нужно перенаправить в `&-`.

Выглядит это так:

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

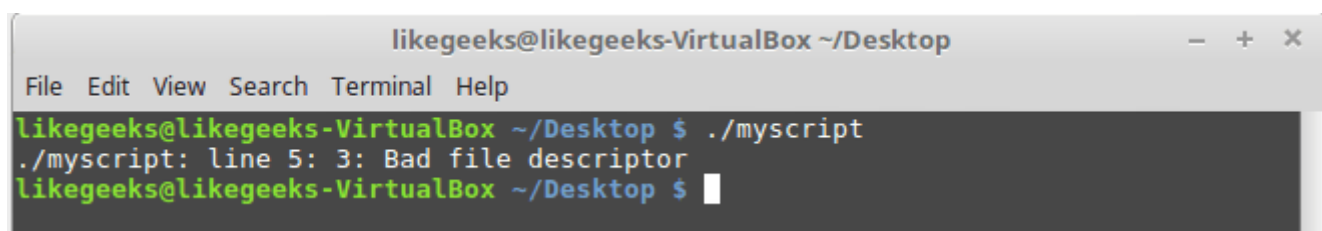
```
exec 3> myfile
```

```
echo "This is a test line of data" >&3
```

```
exec 3>&-
```

```
echo "This won't work" >&3
```

После исполнения скрипта мы получим сообщение об ошибке.

A screenshot of a terminal window titled 'likegeeks@likegeeks-VirtualBox ~/Desktop'. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the command './myscript' being executed. The output is './myscript: line 5: 3: Bad file descriptor'. The prompt returns to 'likegeeks@likegeeks-VirtualBox ~/Desktop \$' with a cursor.

Попытка обращения к закрытому дескриптору файла

Всё дело в том, что мы попытались обратиться к несуществующему дескриптору.

Будьте внимательны, закрывая дескрипторы файлов в сценариях. Если вы отправляли данные в файл, потом закрыли дескриптор, потом — открыли снова, оболочка заменит существующий файл новым. То есть всё то, что было записано в этот файл ранее, будет утеряно.

Получение сведений об открытых дескрипторах

Для того, чтобы получить список всех открытых в Linux дескрипторов, можно воспользоваться командой `lsdf`. Во многих дистрибутивах, вроде Fedora, утилита `lsdf` находится в `/usr/sbin`. Эта команда весьма полезна, так как она выводит сведения о каждом дескрипторе, открытом в системе. Сюда входит и то, что открыли процессы, выполняемые в фоне, и то, что открыто пользователями, вошедшими в систему.

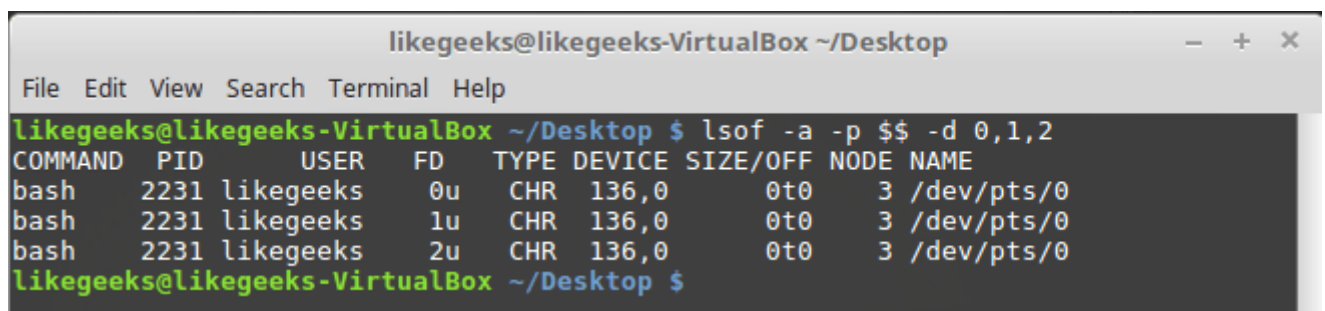
У этой команды есть множество ключей, рассмотрим самые важные.

- -p Позволяет указать ID процесса.
- -d Позволяет указать номер дескриптора, о котором надо получить сведения.

Для того, чтобы узнать PID текущего процесса, можно использовать специальную переменную окружения \$\$, в которую оболочка записывает текущий PID.

Ключ -a используется для выполнения операции логического И над результатами, возвращёнными благодаря использованию двух других ключей:

```
lsof -a -p $$ -d 0,1,2
```



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ lsof -a -p $$ -d 0,1,2
COMMAND PID USER FD TYPE DEVICE SIZE/OFF NODE NAME
bash 2231 likegeeks 0u CHR 136,0 0t0 3 /dev/pts/0
bash 2231 likegeeks 1u CHR 136,0 0t0 3 /dev/pts/0
bash 2231 likegeeks 2u CHR 136,0 0t0 3 /dev/pts/0
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Вывод сведений об открытых дескрипторах

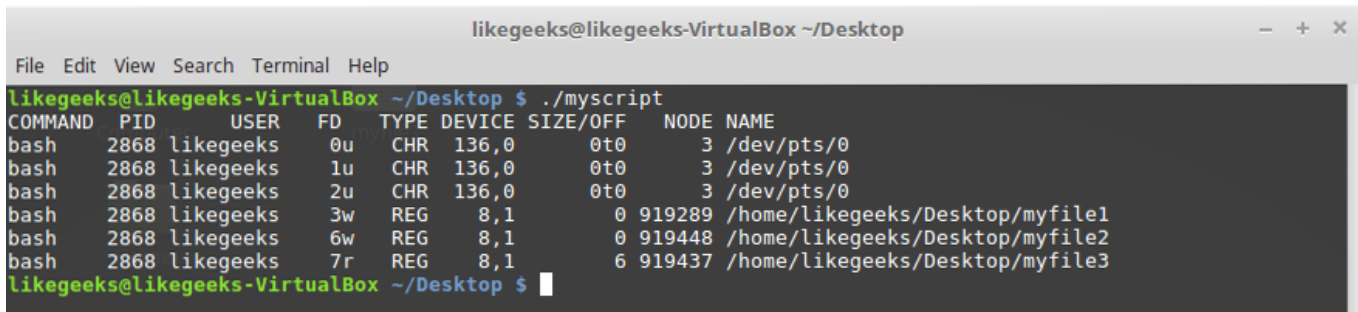
Тип файлов, связанных с STDIN, STDOUT и STDERR — CHR (character mode, символьный режим). Так как все они указывают на терминал, имя файла соответствует имени устройства, назначенного терминалу. Все три стандартных файла доступны и для чтения, и для записи.

Посмотрим на вызов команды lsof из скрипта, в котором открыты, в дополнение к стандартным, другие дескрипторы:

```
#!/bin/bash
exec 3> myfile1
exec 6> myfile2
exec 7< myfile3
```

```
ls -a -p $$ -d 0,1,2,3,6,7
```

Вот что получится, если этот скрипт запустить.



```
likegeeks@likegeeks-VirtualBox ~/Desktop
File Edit View Search Terminal Help
likegeeks@likegeeks-VirtualBox ~/Desktop $ ./myscript
COMMAND  PID    USER    FD    TYPE  DEVICE  SIZE/OFF  NODE  NAME
bash     2868  likegeeks  0u    CHR   136,0    0t0      3    /dev/pts/0
bash     2868  likegeeks  1u    CHR   136,0    0t0      3    /dev/pts/0
bash     2868  likegeeks  2u    CHR   136,0    0t0      3    /dev/pts/0
bash     2868  likegeeks  3w    REG    8,1      0  919289  /home/likegeeks/Desktop/myfile1
bash     2868  likegeeks  6w    REG    8,1      0  919448  /home/likegeeks/Desktop/myfile2
bash     2868  likegeeks  7r    REG    8,1      6  919437  /home/likegeeks/Desktop/myfile3
likegeeks@likegeeks-VirtualBox ~/Desktop $
```

Просмотр дескрипторов файлов, открытых скриптом

Скрипт открыл два дескриптора для вывода (3 и 6) и один — для ввода (7). Тут же показаны и пути к файлам, использованных для настройки дескрипторов.

Подавление вывода

Иногда надо сделать так, чтобы команды в скрипте, который, например, может исполняться как фоновый процесс, ничего не выводили на экран. Для этого можно перенаправить вывод в `/dev/null`. Это — что-то вроде «чёрной дыры».

Вот, например, как подавить вывод сообщений об ошибках:

```
ls -al badfile anotherfile 2> /dev/null
```

Тот же подход используется, если, например, надо очистить файл, не удаляя его:

```
cat /dev/null > myfile
```