Глава 16. Перенаправление ввода/вывода

В системе по-умолчанию всегда открыты три "файла" -- `stdin` (клавиатура), `stdout` (экран) и `stderr` (вывод сообщений об ошибках на экран). Эти, и любые другие открытые файлы, могут быть перенапрвлены. В данном случае, термин "перенаправление" означает получить вывод из файла, команды, программы, сценария или даже отдельного блока в сценарии (см. Пример 3-1 и Пример 3-2) и передать его на вход в другой файл, команду, программу или сценарий.

С каждым открытым файлом связан дескриптор файла. [1] Дескрипторы файлов `stdin`, `stdout` и `stderr` -- 0, 1 и 2, соответственно. При открытии дополнительных файлов, дескрипторы с 3 по 9 остаются незанятыми. Иногда дополнительные дескрипторы могут сослужить неплохую службу, временно сохраняя в себе ссылку на `stdin`, `stdout` или `stderr`. [2] Это упрощает возврат дескрипторов в нормальное состояние после сложных манипуляций с перенаправлением и перестановками (см. Пример 16-1).

```
   COMMAND_OUTPUT >
      # Перенаправление stdout (вывода) в файл.
      # Если файл отсутствовал, то он создется, иначе -- перезаписывается.

      ls -lR > dir-tree.list
      # Создает файл, содержащий список дерева каталогов.

   : > filename
      # Операция > усекает файл "filename" до нулевой длины.
      # Если до выполнения операции файла не существовало,
      # то создается новый файл с нулевой длиной (тот же эффект дает команда
'touch').
      # Символ : выступает здесь в роли местозаполнителя, не выводя ничего.

   > filename
      # Операция > усекает файл "filename" до нулевой длины.
      # Если до выполнения операции файла не существовало,
      # то создается новый файл с нулевой длиной (тот же эффект дает команда
'touch').
      # (тот же результат, что и выше -- ": >", но этот вариант
неработоспособен
      # в некоторых командных оболочках.)

   COMMAND_OUTPUT >>
      # Перенаправление stdout (вывода) в файл.
      # Создает новый файл, если он отсутствовал, иначе -- дописывает в конец
файла.


      # Однострочные команды перенаправления
      # (затрагивают только ту строку, в которой они встречаются):
```

```
      # ------------------------------------------------------------------

   1>filename
      # Перенаправление вывода (stdout) в файл "filename".
   1>>filename
      # Перенаправление вывода (stdout) в файл "filename", файл открывается в
режиме добавления.
   2>filename
      # Перенаправление stderr в файл "filename".
   2>>filename
      # Перенаправление stderr в файл "filename", файл открывается в режиме
добавления.
   &>filename
      # Перенаправление stdout и stderr в файл "filename".


#============================================================================

      # Перенаправление stdout, только для одной строки.
      LOGFILE=script.log

      echo "Эта строка будет записана в файл \"$LOGFILE\"." 1>$LOGFILE
      echo "Эта строка будет добавлена в конец файла \"$LOGFILE\"."
1>>$LOGFILE
      echo "Эта строка тоже будет добавлена в конец файла \"$LOGFILE\"."
1>>$LOGFILE
      echo "Эта строка будет выведена на экран и не попадет в файл
\"$LOGFILE\"."
      # После каждой строки, сделанное перенаправление автоматически
"сбрасывается".



      # Перенаправление stderr, только для одной строки.
      ERRORFILE=script.errors

      bad_command1 2>$ERRORFILE       #  Сообщение об ошибке запишется в
$ERRORFILE.
      bad_command2 2>>$ERRORFILE      #  Сообщение об ошибке добавится в конец
$ERRORFILE.
      bad_command3                    #  Сообщение об ошибке будет выведено на
stderr,
                                      #+ и не попадет в $ERRORFILE.
      # После каждой строки, сделанное перенаправление также автоматически
"сбрасывается".

#============================================================================




   2>&1
      # Перенаправляется stderr на stdout.
      # Сообщения об ошибках передаются туда же, куда и стандартный вывод.

   i>&j
      # Перенаправляется файл с дескриптором i в j.
      # Вывод в файл с дескриптором i передается в файл с дескриптором j.

   >&j
      # Перенаправляется  файл с дескриптором 1 (stdout) в файл с дескриптором
```

```
j.
      # Вывод на stdout передается в файл с дескриптором j.

   0< FILENAME
    < FILENAME
      # Ввод из файла.
      # Парная команде ">", часто встречается в комбинации с ней.
      #
      # grep search-word <filename


   [j]<>filename
      # Файл "filename" открывается на чтение и запись, и связывается с
дескриптором "j".
      # Если "filename" отсутствует, то он создается.
      # Если дескриптор "j" не указан, то, по-умолчанию, бередся дескриптор 0,
stdin.
      #
      # Как одно из применений этого -- запись в конкретную позицию в файле.
      echo 1234567890 > File    # Записать строку в файл "File".
      exec 3<> File       # Открыть "File" и связать с дескриптором 3.
      read -n 4 <&3             # Прочитать 4 символа.
      echo -n . >&3             # Записать символ точки.
      exec 3>&-                 # Закрыть дескриптор 3.
      cat File                  # ==> 1234.67890
      # Произвольный доступ, да и только!



   |
      # Конвейер (канал).
      # Универсальное средство для объединения команд в одну цепочку.
      # Похоже на ">", но на самом деле -- более обширная.
      # Используется для объединения команд, сценариев, файлов и программ в
одну цепочку (конвейер).
      cat *.txt | sort | uniq > result-file
      # Содержимое всех файлов .txt сортируется, удаляются повторяющиеся
строки,
      # результат сохраняется в файле "result-file".
```

Операции перенаправления и/или конвейеры могут комбинироваться в одной командной строке.

```
command < input-file > output-file

command1 | command2 | command3 > output-file
```

См. Пример 12-23 и Пример A-17.

Допускается перенаправление нескольких потоков в один файл.

```
ls -yz >> command.log 2>&1
# Сообщение о неверной опции "yz" в команде "ls" будет записано в файл
"command.log".
# Поскольку stderr перенаправлен в файл.
```

Закрытие дескрипторов файлов

n<&-

Закрыть дескриптор входного файла *n*.

0<&-, <&-

Закрыть `stdin`.

n>&-

Закрыть дескриптор выходного файла *n*.

1>&-, >&-

Закрыть `stdout`.

Дочерние процессы наследуют дескрипторы открытых файлов. По этой причине и работают конвейеры. Чтобы предотвратить наследование дескрипторов -- закройте их перед запуском дочернего процесса.

```
# В конвейер передается только stderr.

exec 3>&1                                # Сохранить текущее "состояние" stdout.
ls -l 2>&1 >&3 3>&- | grep bad 3>&-    # Закрыть дескр. 3 для 'grep' (но не
для 'ls').
#                    ^^^^   ^^^^
exec 3>&-                                # Теперь закрыть его для оставшейся
части сценария.

# Спасибо S.C.
```

Дополнительные сведения о перенаправлении ввода/вывода вы найдете в Приложение D.

16.1. С помощью команды exec

Команда **exec <filename** перенаправляет ввод со `stdin` на файл. С этого момента весь ввод, вместо `stdin` (обычно это клавиатура), будет производиться из этого файла. Это дает возможность читать содержимое файла, строку за строкой, и анализировать каждую введенную строку с помощью sed и/или awk.

Пример 16-1. Перенаправление `stdin` с помощью exec

```
#!/bin/bash
# Перенаправление stdin с помощью 'exec'.


exec 6<&0          # Связать дескр. #6 со стандартным вводом (stdin).
                   # Сохраняя stdin.

exec < data-file   # stdin заменяется файлом "data-file"

read a1            # Читается первая строка из "data-file".
read a2            # Читается вторая строка из "data-file."
```

```
echo
echo "Следующие строки были прочитаны из файла."
echo "-----------------------------------------"
echo $a1
echo $a2

echo; echo; echo

exec 0<&6 6<&-
#  Восстанавливается stdin из дескр. #6, где он был предварительно сохранен,
#+ и дескр. #6 закрывается ( 6<&- ) освобождая его для других процессов.
#
# <&6 6<&-    дает тот же результат.

echo -n "Введите строку  "
read b1  # Теперь функция "read", как и следовало ожидать, принимает данные с
обычного stdin.
echo "Строка, принятая со stdin."
echo "--------------------------"
echo "b1 = $b1"

echo

exit 0
```

Аналогично, конструкция **exec >filename** перенаправляет вывод на `stdout` в заданный файл. После этого, весь вывод от команд, который обычно направляется на `stdout`, теперь выводится в этот файл.

Пример 16-2. Перенаправление `stdout` с помощью exec

```
#!/bin/bash
# reassign-stdout.sh

LOGFILE=logfile.txt

exec 6>&1           # Связать дескр. #6 со stdout.
                    # Сохраняя stdout.

exec > $LOGFILE     # stdout замещается файлом "logfile.txt".

# ---------------------------------------------------------- #
# Весь вывод от команд, в данном блоке, записывается в файл $LOGFILE.

echo -n "Logfile: "
date
echo "-----------------------------------"
echo

echo "Вывод команды \"ls -al\""
echo
ls -al
echo; echo
echo "Вывод команды \"df\""
echo
df

# ---------------------------------------------------------- #

exec 1>&6 6>&-      # Восстановить stdout и закрыть дескр. #6.
```

```
echo
echo "== stdout восстановлено в значение по-умолчанию == "
echo
ls -al
echo

exit 0
```

Пример 16-3. Одновременное перенаправление устройств, `stdin` и `stdout`, с помощью команды exec

```
#!/bin/bash
# upperconv.sh
# Преобразование символов во входном файле в верхний регистр.

E_FILE_ACCESS=70
E_WRONG_ARGS=71

if [ ! -r "$1" ]     # Файл доступен для чтения?
then
  echo "Невозможно прочитать из заданного файла!"
  echo "Порядок использования: $0 input-file output-file"
  exit $E_FILE_ACCESS
fi                   #  В случае, если входной файл ($1) не задан
                     #+ код завершения будет этим же.

if [ -z "$2" ]
then
  echo "Необходимо задать выходной файл."
  echo "Порядок использования: $0 input-file output-file"
  exit $E_WRONG_ARGS
fi


exec 4<&0
exec < $1            # Назначить ввод из входного файла.

exec 7>&1
exec > $2            # Назначить вывод в выходной файл.
                     # Предполагается, что выходной файл доступен для записи
                     # (добавить проверку?).

# -----------------------------------------------
    cat - | tr a-z A-Z   # Перевод в верхний регистр
#   ^^^^^               # Чтение со stdin.
#           ^^^^^^^^^^  # Запись в stdout.
# Однако, и stdin и stdout были перенаправлены.
# -----------------------------------------------

exec 1>&7 7>&-       # Восстановить stdout.
exec 0<&4 4<&-       # Восстановить stdin.

# После восстановления, следующая строка выводится на stdout, чего и следовало
ожидать.
echo "Символы из \"$1\" преобразованы в верхний регистр, результат записан в
\"$2\"."

exit 0
```

Примечания

[1] *дескриптор файла* -- это просто число, по которому система идентифицирует открытые файлы. Рассматривайте его как упрощенную версию указателя на файл.

[2] При использрвании *дескриптора с номером 5* могут возникать проблемы. Когда Bash порождает дочерний процесс, например командой exec, то дочерний процесс наследует дескриптор 5 как "открытый" (см. архив почты Чета Рамея (Chet Ramey), SUBJECT: RE: File descriptor 5 is held open) Поэтому, лучше не использовать этот дескриптор.

Назад
Арифметические подстановки

К началу
Наверх

Вперед
Перенаправление для блоков кода