Программные каналы и команды

Перенаправление потоков ввода/вы-вода

Одной из мощных возможностей командной оболочки системы Unix является механизм **перенаправления потоков ввода/вывода** с возможностью задействования программных **каналов**.

В данной главе даются пояснения относительно **перенаправления** стандартных потоков ввода, вывода и ошибок.

Потоки данных stdin, stdout и stderr

Командная оболочка bash поддерживает три типа базовых потоков данных; она принимает данные из стандартного потока ввода **stdin** (поток **0**), отправляет данные в стандартный поток вывода **stdout** (поток **1**), а также отправляет сообщения об ошибках в стандартный поток ошибок **stderr** (поток **2**).

Приведенная ниже иллюстрация является графической интерпретацией этих трех потоков данных.

Клавиатура обычно служит источником данных для стандартного потока ввода **stdin**, в то время, как стандартные потоки вывода **stdout** и ошибок **stderr** используются для вывода данных. Новых пользователей Linux может смущать подобное разделение, так как не существует очевидного способа дифференцирования стандартных потоков вывода **stdout** и ошибок **stderr**. Опытные же пользователи знают о том, что разделение стандартных потоков вывода и ошибок может оказаться весьма полезным.

В следующем разделе будет рассказано о том, как осуществляется перенаправление упомянутых потоков данных.

Перенаправление стандартного потока вывода

Операция перенаправления потока данных stdout (>)

Перенаправление стандартного потока вывода **stdout** может быть осуществлено с помощью символа знака "**больше**". В том случае, если при разборе строки команды командная оболочка обнаруживает символ знака **>**, она удаляет данные из файла и перенаправлет данные из стандартного потока вывода в него.

Нотация **>** фактически является аббревиатурой для **1>** (в данном случае стандартный поток вывода обозначается как поток номер **1**).

```
[paul@RHELv4u3 ~]$ echo Сегодня холодно!
Сегодня холодно!
[paul@RHELv4u3 ~]$ echo Сегодня холодно! > winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
Сегодня холодно!
[paul@RHELv4u3 ~]$
```

Обратите внимание на то, что командная оболочка bash фактически **удаляет** описание операции перенаправления потока данных из строки команды перед исполнением этой команды, представленной аргументом 0. Это значит, что в случае исполнения данной команды:

```
echo привет > greetings.txt
```

командная оболочка будет рассматривать только два аргумента (echo = аргумент 0, привет = аргумент 1). Описание операции перенаправления потока данных удаляется перед началом подсчета количества аргументов.

Содержимое выходного файла удаляется

В том случае, если в процессе разбора строки команды командная оболочка обнаружит символ знака **>**, содержимое указанного после него файла **будет удалено!** Ввиду того, что описанная процедура

выполняется перед извлечением **аргумента 0**, содержимое файла будет удалено даже в случае неудачного исполнения команды!

```
[paul@RHELv4u3 ~]$ cat winter.txt
Сегодня холодно!
[paul@RHELv4u3 ~]$ zcho Сегодня холодно! > winter.txt
-bash: zcho: команда не найдена..
[paul@RHELv4u3 ~]$ cat winter.txt
[paul@RHELv4u3 ~]$
```

Параметр командной оболочки noclobber

Удаление содержимого файла при использовании оператора **>** может быть предотвращено путем установки параметра командной оболочки **noclobber**.

```
[paul@RHELv4u3 ~]$ cat winter.txt
Сегодня холодно!
[paul@RHELv4u3 ~]$ set -o noclobber
[paul@RHELv4u3 ~]$ echo Сегодня холодно! > winter.txt
-bash: winter.txt: не могу переписать уже существующий файл
[paul@RHELv4u3 ~]$ set +o noclobber
[paul@RHELv4u3 ~]$
```

Нейтрализация влияния параметра командной оболочки noclobber

Влияние параметра командной оболочки **noclobber** может быть нейтрализовано с помощью оператора **>|**.

```
[paul@RHELv4u3 ~]$ set -o noclobber
[paul@RHELv4u3 ~]$ echo Сегодня холодно! > winter.txt
-bash: winter.txt: не могу переписать уже существующий файл
[paul@RHELv4u3 ~]$ echo Сегодня очень холодно! >| winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
Сегодня очень холодно!
[paul@RHELv4u3 ~]$
```

Оператор дополнения >>

Следует использовать оператор **>>** для **записи** данных из стандартного потока вывода в конец файла без предварительного удаления содержимого этого файла.

```
[paul@RHELv4u3 ~]$ echo Сегодня холодно! > winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
Сегодня холодно!
[paul@RHELv4u3 ~]$ echo Когда же наступит лето ? >> winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
Сегодня холодно!
Когда же наступит лето ?
[paul@RHELv4u3 ~]$
```

Перенаправление стандартного потока ошибок

Операция перенаправления потока данных stderr (2>)

Перенаправление стандартного потока ошибок осуществляется с помощью оператора **2>**. Такое перенаправление может оказаться очень полезным для предотвращения заполнения вашего экрана сообщениями об ошибках.

В примере ниже показана методика перенаправления данных из стандартного потока вывода в файл, а данных из стандартного потока ошибок - в специальный файл устройства **/dev/null**. Запись **1>** идентична записи **>**.

```
[paul@RHELv4u3 ~]$ find / > allfiles.txt 2> /dev/null
[paul@RHELv4u3 ~]$
```

Операция перенаправления нескольких потоков данных 2>&1

Для перенаправления данных как из стандартного потока вывода, так и из стандартного потока ошибок в один и тот же файл следует использовать конструкцию **2>&1**.

```
[paul@RHELv4u3 ~]$ find / > allfiles_and_errors.txt 2>&1
[paul@RHELv4u3 ~]$
```

Помните о том, что последовательность операций перенаправления потоков данных имеет значение. К примеру, команда

```
ls > dirlist 2>&1
```

позволяет перенаправить как данные из стандартного потока вывода (с файловым дескриптором 1), так и данные из стандартного потока ошибок (с файловым дескриптором 2) в файл `dirlist`, в то время, как команда

```
ls 2>&1 > dirlist
```

позволяет перенаправить только данные из стандартного потока вывода в файл dirlist, так как с помощью данной команды осуществляется копирование дескриптора стандартного потока вывода в дескриптор стандартного потока ошибок перед тем, как стандартный поток вывода перенаправляется в файл `dirlist`.

Перенаправление стандартного потока вывода и программные каналы

По умолчанию вы не можете использовать утилиту `grep` для обработки данных стандартного потока ошибок **stderr** приложения при использовании программных каналов в рамках строки команды, так как данная утилита получает данные исключительно из стандартного потока вывода **stdout** приложения.

```
paul@debian7:~$ rm file42 file33 file1201 | grep file42
rm: невозможно удалить "file42": Нет такого файла или каталога
rm: невозможно удалить "file33": Нет такого файла или каталога
rm: невозможно удалить "file1201": Нет такого файла или каталога
```

С помощью конструкции **2>&1** вы можете переправить данные из стандартного потока ошибок **stderr** в стандартный поток вывода **stdout** приложения. Это обстоятельство позволяет обрабатывать пере-

даваемые посредством программного канала данные из обоих потоков с помощью следующей команды.

```
paul@debian7:~$ rm file42 file33 file1201 2>&1 | grep
file42
rm: невозможно удалить "file42": Нет такого файла или ка-
талога
```

Вы не можете одновременно использовать конструкции **1>&2** и **2>&1** для осуществления обмена файловых дескрипторов между стандартным потоком вывода **stdout** и стандартным потоком ошибок **stderr**.

```
paul@debian7:~$ rm file42 file33 file1201 2>&1 1>&2 |
grep file42
rm: невозможно удалить "file42": Нет такого файла или ка-
талога
paul@debian7:~$ echo file42 2>&1 1>&2 | sed
's/file42/FILE42/'
FILE42
```

Вам потребуется третий поток данных для осуществления обмена файловых дескрипторов между стандартным потоком вывода **stdout** и стандартным потоком ошибок **stderr** перед символом для создания программного канала.

```
paul@debian7:~$ echo file42 3>&1 1>&2 2>&3 | sed
's/file42/FILE42/'
file42
paul@debian7:~$ rm file42 3>&1 1>&2 2>&3 | sed
's/file42/FILE42/'
rm: невозможно удалить "FILE42": Нет такого файла или ка-
талога
```

Объединение стандартных потоков вывода stdout и ошибок stderr

Конструкция **&>** позволяет объединить стандартные потоки вывода **stdout** и ошибок **stderr** в рамках одного потока данных (причем данные будут сохраняться в файле).

```
paul@debian7:~$ rm file42 &> out_and_err
paul@debian7:~$ cat out_and_err
rm: невозможно удалить "file42": Нет такого файла или ка-
талога
```

```
paul@debian7:~$ echo file42 &> out_and_err
paul@debian7:~$ cat out_and_err
file42
paul@debian7:~$
```

Перенаправление стандартного потока ввода

Операция перенаправления потока данных stdin (<)

Перенаправление стандартного потока ввода **stdin** осуществляется с помощью оператора **<** (являющегося краткой версией оператора **0<**).

```
[paul@RHEL4b ~]$ cat < text.txt
one
two
[paul@RHEL4b ~]$ tr 'onetw' 'ONEZZ' < text.txt
ONE
ZZO
[paul@RHEL4b ~]$
```

Структура << here document

Структура **here document** (иногда называемая структурой here-is-document) является механизмом для ввода данных до момента обнаружения определенной последовательности символов (обычно EOF). Маркер **EOF** может быть либо введен вручную, либо вставлен автоматически при нажатии комбинации клавиш Ctrl-D.

```
[paul@RHEL4b ~]$ cat < text.txt
> один
> два
> EOF
[paul@RHEL4b ~]$ cat text.txt
один
два
[paul@RHEL4b ~]$ cat < text.txt
> brel
> brol
[paul@RHEL4b ~]$ cat text.txt
brel
```

```
[paul@RHEL4b ~]$
```

Структура <<< here string

Структура **here string** может использоваться для непосредственной передачи строк команде. При использовании данной структуры достигается такой же эффект, как и при использовании команды **echo строка | команда** (но вы сможете избежать создания одного дополнительного процесса).

```
paul@ubu1110~$ base64 <<< linux-training.be
bGludXgtdHJhaW5pbmcuYmUK
paul@ubu1110~$ base64 -d <<< bGludXgtdHJhaW5pbmcuYmUK
linux-training.be
```

Для получения дополнительной информации об алгоритме **base64** следует обратиться к стандарту **rfc 3548**.

Неоднозначное перенаправление потоков ввода/вывода

Командная оболочка будет осуществлять разбор всей строки команды перед осуществлением перенаправления потоков ввода/вывода. Следующая команда является хорошо читаемой и корректной:

```
cat winter.txt > snow.txt 2> errors.txt
```

Но следующая команды также является корректной, хотя и хуже читается:

```
2> errors.txt cat winter.txt > snow.txt
```

Даже следующая команда будет прекрасно интерпретироваться командной оболочкой:

```
< winter.txt > snow.txt 2> errors.txt cat
```

Быстрая очистка содержимого файла

Так какой же самый быстрый способ очистки содержимого файла?

```
>foo
```

А какой самый быстрый способ очистки содержимого файла в случае активации параметра командной оболочки **noclobber**?

```
>|bar
```