Стандартные потоки ввода/вывода.

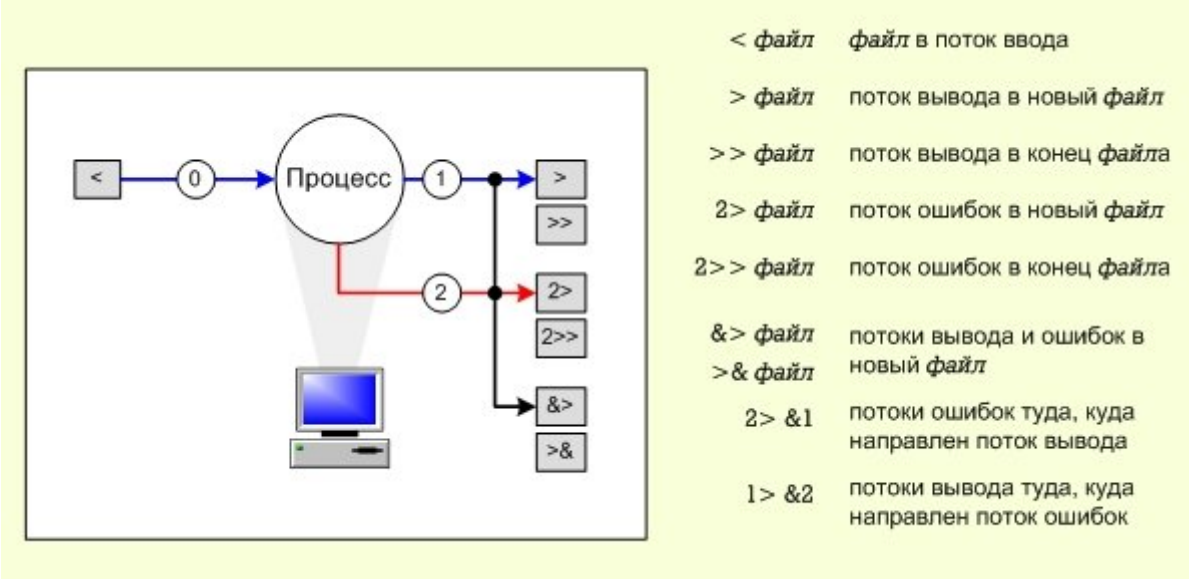
Стандартные потоки ввода и вывода в UNIX/Linux наряду с файлами являются одним из наиболее распространённых средств для обмена информацией процессов с внешним миром, а перенаправления >, >> и |, одной из самых популярных конструкций командного интерпретатора.

На этой странице рассматриваются как базовые вопросы использования потоков ввода/вывода, так и тонкости и хитрости, например, почему не работает echo text | read ver и многие другие.

Содержание

1. Потоки и файлы.

Процесс взаимодействия с пользователем выполняется в терминах записи и чтения в файл. То есть вывод на экран представляется как запись в файл, а ввод — как чтение файла. Файл, из которого осуществляется чтение, называется *стандартным потоком ввода*, а в который осуществляется запись — *стандартным потоком вывода*.



Стандартные потоки — воображаемые файлы, позволяющие осуществлять взаимодействие с пользователем как чтение и запись в файл. Кроме потоков ввода и вывода, существует еще и *стандартный поток ошибок*, на который выводятся все сообщения об ошибках и те информативные сообщения о ходе работы программы, которые не могут быть выведены в стандартный поток вывода.

Стандартные потоки привязаны к файловым дескрипторам с номерами 0, 1 и 2.

- Стандартный поток ввода (stdin) — 0;
- Стандартный поток вывода (stdout) — 1;
- Стандартный поток ошибок (stderr) — 2.

Вывод данных на экран и чтение их с клавиатуры происходит потому, что по умолчанию стандартные потоки ассоциированы с терминалом пользователя. Это не является обязательным — потоки можно подключать к чему угодно — к файлам, программам и даже устройствам. В командном интерпретаторе bash такая операция называется *перенаправлением*.

< файл

Использовать файл как источник данных для стандартного потока ввода.

> файл

Направить стандартный поток вывода в файл. Если файл не существует, он будет создан; если существует — перезаписан сверху.

2> файл

Направить стандартный поток ошибок в файл. Если файл не существует, он будет создан; если существует — перезаписан сверху.

>>файл

Направить стандартный поток вывода в файл. Если файл не существует, он будет создан; если существует — данные будут дописаны к нему в конец.

2>>файл

Направить стандартный поток ошибок в файл. Если файл не существует, он будет создан; если существует — данные будут дописаны к нему в конец.

&>файл или >&файл

Направить стандартный поток вывода и стандартный поток ошибок в файл. Другая форма записи: >файл 2>&1.

>&-

Закрыть поток вывода перед вызовом команды;

2>&-

Закрыть поток ошибок перед вызовом команды;

cat <<EOF

Весь текст между блоками EOF (в общем случае вместо EOF можно использовать любое слово) будет выведен на экран. Важно: перед последним EOF не должно быть пробелов! (heredoc синтаксис).

EOF

<<<string

Аналогично, но только для одной строки (для bash версии 3 и выше)

Пример. Эта команда объединяет три файла: header, body и footer в один файл letter:

```
%$ cat header body footer > letter
```

Команда cat по очереди выводит содержимое файлов, перечисленных в качестве параметров на стандартный поток вывода. Стандартный поток вывода перенаправлен в файл letter.

Здесь используется сразу перенаправление стандартного потока ввода и стандартного потока вывода:

```
%$ sort < unsortedlines > sortedlines
```

Программа **sort** сортирует данные, поступившие в стандартный поток ввода, и выводит их на стандартный поток вывода. Стандартный поток ввода подключен к файлу unsortedlines, а выход записывается в sortedlines.

Здесь перенаправлены потоки вывода и ошибок:

```
%$ find /home -name '*.rpm' >rpmlist 2> /dev/null
```

Программа **find** ищет в каталоге /home файлы с суффиксом .rpm. Список найденных файлов записывается в файл rpmlist. Все сообщения об ошибках удаляются. Удаление достигается при помощи перенаправления потока ошибок на устройство /dev/null — специальный файл, означающий ничто. Данные, отправленные туда, безвозвратно исчезают. Если же прочитать содержимое этого файла, он окажется пуст.

Для того чтобы лучше понять, что потоки работают как файлы, рассмотрим такой пример:

```
%$ cat > /tmp/fff &

[1] 28378

[1]+  Stopped                 cat > /tmp/fff

%$ ls -l /proc/28378/fd/

total 0

lrwx------ 1 igor igor 64 2009-06-24 18:58 0 -> /dev/pts/1

l-wx------ 1 igor igor 64 2009-06-24 18:58 1 -> /tmp/fff

lrwx------ 1 igor igor 64 2009-06-24 18:58 2 -> /dev/pts/1

l-wx------ 1 igor igor 64 2009-06-24 18:58 5 -> pipe:[13325]

lr-x------ 1 igor igor 64 2009-06-24 18:58 7 -> pipe:[13329]
```

Программа **cat** запускается для записи данных в файл /tmp/fff. Он запускается в фоне (&), и получает номер работы 1 ([1]). Процесс этой программы имеет номер 28378.

Информация о процессе 28738 находится в каталоге /proc/28738 специальной псевдофайловой системы /proc. В частности, в подкаталоге /proc/28738/fd/ находится список файловых дескрипторов для открытых процессом файлов.

Здесь видно, что стандартный поток ввода (0), и стандартный поток ошибок (2) процесса подключены на терминал, а вот стандартный поток вывода (1)

перенаправлен в файл.

Завершить работу программы **cat** можно командой kill %1.

Командный интерпретатор — это тоже процесс. И у него есть стандартные потоки ввода и вывода. Если интерпретатор работает в интерактивном режиме, то они подключены на консоль (вывода на экран; чтение с клавиатуры). Можно обратиться напрямую к этим потокам изнутри интерпретатора:

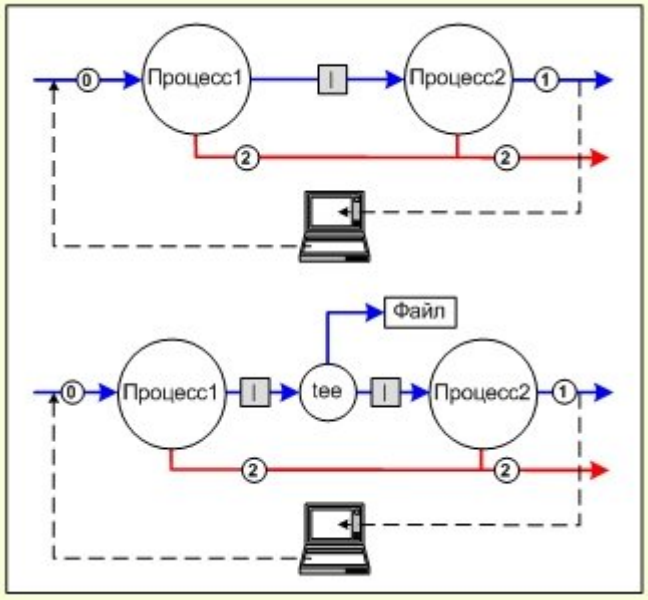
- /dev/stdin — стандартный поток ввода;
- /dev/stdout — стандартный поток вывода;
- /dev/stderr — стандартный поток ошибок.

Например, здесь видно, что потоки ссылаются в конечном итоге на файл-устройство терминала, с которым работает интерпретатор:

```
%$ ls -l /dev/std*

lrwxrwxrwx 1 root root 15 2009-06-17 23:54 /dev/stderr -> /proc/self/fd/2

lrwxrwxrwx 1 root root 15 2009-06-17 23:54 /dev/stdin -> /proc/self/fd/0

lrwxrwxrwx 1 root root 15 2009-06-17 23:54 /dev/stdout -> /proc/self/fd/1

%$ ls -l /proc/self/fd/[012]

lrwx------ 1 igor igor 64 2009-06-24 18:16 /proc/self/fd/0 -> /dev/pts/1

lrwx------ 1 igor igor 64 2009-06-24 18:16 /proc/self/fd/1 -> /dev/pts/1

lrwx------ 1 igor igor 64 2009-06-24 18:16 /proc/self/fd/2 -> /dev/pts/1

%$ tty

/dev/pts/1
```

2. Каналы.

Стандартные потоки можно перенаправлять не только в файлы, но и на вход других программ. Если поток вывода одной программы соединить с потоком ввода другой программы, получится конструкция, называемая каналом, конвейером или пайпом (от англ. pipe, труба).



команда1 | команда2

Стандартный поток вывода **команды1** подключается к стандартному потоку ввода **команды2**

команда1 | tee *файл* | команда2

Данные, передаваемые со стандартного потока вывода **команды1** на стандартный поток ввода **команды2**, сохраняются в ***файл***

В **bash** канал выглядит как последовательность команд, отделенных друг от друга символом |:

команда1 | *команда2* | *команда3* ...

Стандартный поток вывода *команды1* подключается к стандартному потоку ввода *команды2*, стандартный поток вывода *команды2* в свою очередь подключается к потоку ввода *команды3* и т.д.

В UNIX/Linux существует целый класс команд, предназначенных для преобразования потоков данных в каналах. Такие программы известны как *фильтры*. Программа-фильтр читает данные, поступающие со стандартного потока ввода (на вход), преобразовывает их требуемым образом и выводит на стандартный поток вывода (на выход). Существует множество хорошо известных фильтров, призванных решать определенные задачи, и являющихся незаменимым инструментом в руках пользователя ОС.

Каналы в ОС Linux являются одной из наиболее часто применяемых конструкций, а фильтры — наиболее часто применяемых программ. Большинство повседневных задач в Linux легко решаются при помощи конструкций построенных на основе нескольких фильтров.

Программы, образующие канал, выполняются параллельно как независимые процессы.

Можно создавать ответвление в каналах. Команда tee позволяет сохранять

данные, передающиеся в канале:

tee [*опции*] *файл*

Программа **tee** копирует данные, поступающие на стандартный поток ввода, в указанные в качестве аргументов команды файлы, и передает данные на стандартный поток вывода.

Рассмотренный ниже пример: сортируется файл unsortedlines и результат записывается в sortedlines.

```
%$ cat unsortedlines | sort > sortedlines
```

Команда выполняет те же действия, но запись является более наглядной.

Вот пример посложнее. Вывести название и размер пользовательского каталога, занимающее наибольшее место на диске.

```
$ du -s /home/* | sort -nr | head -1
```

Программа **du**, при вызове ее с ключом -s, сообщает суммарный объем каждого каталога или файла, перечисленного в ее параметрах.

Ключ -n команды **sort** означает, что сортировка должна быть арифметической, т.е. строки должны рассматриваться как числа, а не как последовательности символов (Например, 12>5 в то время как строка '12'<'5' т.к. сравнение строк производится посимвольно и '1'<'5'). Ключ -r означает изменения порядка сортировки — с возрастающего на убывающий.

Команда **head** выводит несколько первых строк поступающего на ее вход потока, отбрасывая все остальные. Ключ -1 означает, что надо вывести только одну строку.

Таким образом, список пользовательских каталогов с их суммарным объемом арифметически сортируется по убыванию, и из полученного списка берется первая строка, т.е. строка с наибольшим числом, соответствующая самому объемному каталогу.

Использование команды **tee**:

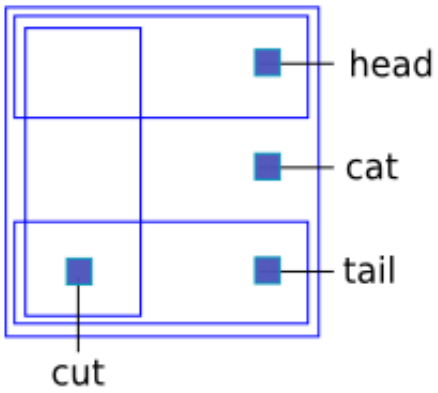
```
$ sort text | tee sorted_text | head -n 1
```

Содержимое файла text сортируется, и результат сортировки записывается в файл sorted_text. Первая строка отсортированного текста выдается на экран.

3. Программы фильтры.

В UNIX/Linux существует целый класс команд, которые принимают данные со стандартного потока ввода, каким-то образом обрабатывают их, и выдают результат на стандартный поток вывода. Такие программы называются программами-фильтрами.

Как правило, все эти программы работают как фильтры, если у них нет аргументов (опции могут быть), но как только им в качестве аргумента передаётся файл, они считывают данные из этого файла, а не со стандартного потока ввода (существуют и исключения, например, программа **tr**, которая обрабатывает данные поступающие исключительно через стандартный поток ввода).



cat

Считывает данные со стандартного потока ввода и передаёт их на стандартный поток вывода. Без опций работает как простой повторитель. С опциями может фильтровать пустые строки, нумеровать строки и делать другую подобную работу.

head

Показывает первые 10 строк (или другое заданное количество), считанных со стандартного потока ввода.

tail

Показывает последние 10 строк (или другое заданное количество), считанные со стандартного потока ввода. Важный частный случай **tail** -f, который в режиме слежения показывает концовку файла. Это используется, в частности, для просмотра файлов журнальных сообщений.

cut

Вырезает столбец (по символам или полям) из потока ввода и передаёт на поток вывода. В качестве разделителей полей могут использоваться любые символы.

sort

Отсортировать данные в соответствии с какими-либо критериями, например, арифметически по второму столбцу.

uniq

Удалить повторяющиеся строки. Или (с ключом -c) не просто удалить, а написать сколько таких строк было. Учитываются только подряд идущие одинаковые строки, поэтому часто данные сортируются перед тем как отправить их на вход программе.

tee

Ответвить данные в файл. Используется для сохранения промежуточных данных, передающихся в потоке, в файл.

bc

Вычислить каждую отдельную строку потока и записать вместо неё результат вычисления.

hexdump

Показать шестнадцатеричное представление данных, поступающих на стандартный поток ввода.

strings

вычленить и показать в стандартном потоке (или файле) то, что напоминает строки. Всё что не похоже на строковые последовательности, игнорировать. Полезна в сочетании с **grep** для поиска интересующих строковых последовательностей в бинарных файлах.

grep

Отфильтровать поток, и показать только строки, содержащие (или не содержащие) заданное регулярное выражение.

tr

Посимвольная замена текста в потоке. Например, **tr** A-Z a-z меняет регистр символов с большого на маленький.

sed

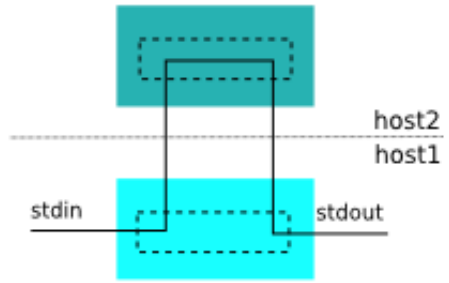
Обработать текст в соответствии с заданным скриптом. Наиболее часто используется для замены текста в потоке: **sed** s/было/стало/g

awk

Обработать текст в соответствии с заданным скриптом. Как правило, используется для обработки текстовых таблиц, например таких как вывод ps aux и тому подобных, но не только.

perl

Обработать текст в соответствии с заданным скриптом. Возможности языка Perl выходят далеко за рамки однострочников для командной строки, но с однострочниками он справляется особенно виртуозно. В Perl существует оператор <> (diamond operator) и конструкция while(<>) { ... }, которая предполагает обработку данных со стандартного потока ввода (или из файлов, если они переданы в качестве аргументов). При написании однострочников можно использовать ключи -n (равносильный оборачиванию кода в while(<>) { ... }) или -p (равносильный while(<>) { ... }).




```
host1$ ssh host2 cmd1
```

sh -s

Текст, который передаётся на стандартный поток ввода **sh -s** может интерпретироваться как последовательность команд shell. На выход передаётся результат их исполнения.

ssh

Средство удалённого доступа ssh, может работать как фильтр. **ssh** подхватывает данные, переданные ему на стандартный поток ввода, передаёт их на удалённых хост и подаёт на вход процессу программы, имя которой было передано ему в качестве аргумента. Результат выполнения программы (то есть то, что она выдала на стандартный поток вывода) передаётся со стандартного вывода ssh.

4. Переменные командного интерпретатора и потоки.

Для того чтобы вывести содержимое переменной командного интерпретатора на стандартный поток вывода, используется команда **echo**:

```
%$ echo $VAR
```

На стандартный поток ошибок данные можно передать с помощью перенаправления:

```
%$ echo $VAR > /dev/stderr
```

Содержимое переменной (или любой другой текст) поступят не на стандартный поток вывода, а на стандартный поток ошибок.

Считать данные со стандартного потока ввода внутрь одной или нескольких переменных можно при помощи команды **read**:

```
%$ read VAR
```

Строка из стандартного потока ввода считается внутрь переменной VAR.

Если указать несколько переменных, то в первую попадёт первое слово; во вторую — второе слово; в последнюю — всё остальное.

```
%$ df -h  | head -1

Filesystem            Size  Used Avail Use% Mounted on

%$ df -h  | head -1 | { read VAR1 VAR2 VAR3; echo $VAR1; echo $VAR2; echo $VAR3 ; }

Filesystem

Size

Used Avail Use% Mounted on
```

Обратите внимание на конструкцию { **read** VAR; **echo** $VAR }. Переменная считанная из канала с помощью **read** будет доступна только внутри { }. Например echo text | read var; echo $var выведет пустое место. Это связано с тем, что для перенаправления внутрь read shell порождает дочерний процесс, в котором и исполняет read. Переменная дочернего процесса не передаётся наружу. (Внимание. В **zsh** всё работает и без этих костылей).

Если прочитать данные со стандартного потока ввода не удалось, то команда **read** возвращает код завершения отличный от нуля. Это позволяет использовать **read**, например, в такой конструкции:

```
 %$ cat users | while read user

   do
```

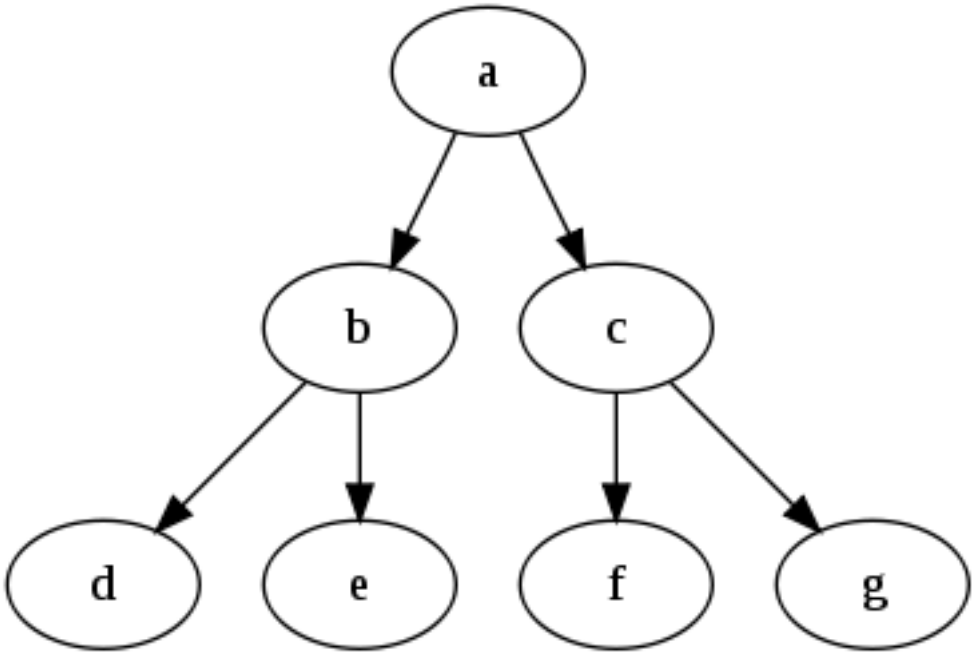
```
    useradd $user

done
```

В этом примере **read** считывает строки из файла users, и для каждой прочитанной строки вызывается команда **useradd**, которая добавляет пользователя в системе. В результате: создаются учётные записи пользователей, имена которых перечислены в файле users.

Переменная user после выхода из цикла остаётся в том же виде, в каком она была до входа в цикл (а не содержит последнее значение файла, как можно было бы предположить). Это связано с тем, что для обработки **while**, на вход которому направлен канал, порождается дочерний интерпретатор, и модификация переменной происходит внутри него.

5. Именованные каналы.

Безымянный канал можно построить только между процессами, которые порождены от одного процесса (и на практике они должны быть порождены одновременно, а не последовательно, хотя теоретически это не обязательно). Если же процессы имеют разных родителей, то между ними обычный, безымянный канал построить не получится.



Например, в данном случае d и e, и f и g легко могут быть соединены при помощи канала, но e и f соединить с помощью канала не получится.

Для решения этой задачи используются именованные каналы *fifo* (first in, first out). Они во всём повторяют обычные каналы (pipe), только имеют привязку к файловой системе. Создать именованный канал можно командой **mkfifo**:

```
%$ mkfifo /tmp/fifo
```

Созданный канал можно использовать для соединения процессов между собой. Например, эти перенаправления будут работать одинаково:

```
%$ f | g
```

и

```
%$ f > /tmp/fifo & g < /tmp/fifo
```

(здесь f и g — процессы из вышеуказанной иерархии процессов). Процессы f и g имеют общего предка. А вот для процессов *e* и *g*, не связанных между собой, обычный канал использовать не получится, только именованный:

```
%$ e > /tmp/fifo&

%$ g < /tmp/fifo   # в другом интерпретаторе
```

6. Потоки ввода/вывода и терминал.

Большинство программ, которые работают с потоками ввода и вывода, работают с ними как с простыми файлами, и не рассчитывают на то, что поток подключен к терминалу. Но не все. Если потоки ввода/вывода отключаются от терминала и перенаправляются в файл, часть возможностей программы может пропасть.

Например, если командный интерпретатор отвязать от терминала, то у него потеряется множество интерактивных возможностей:

```
%$ cat | sh -i | cat
```

Например, возможности прокручивать историю команд у него теперь нет. Возможности по редактированию теперь тоже сильно урезаны. Фактически, редактирование команды теперь выполняется уже с помощью программы **cat**, которая держит терминал, а интерпретатору поступают уже полностью введённые команды.

Проверить, подключен ли наш стандартный поток к терминалу, или он перенаправлен в файл, можно при помощи **test**, ключ -t:

```
%$ test -t 0 && echo terminal

terminal

%$ cat | test -t 0 && echo terminal

%$ cat | test -t 0 || echo file

file
```

7. Подстановка процесса.

Есть хитрый трюк, который в чистом виде перенаправлением потока ввода/вывода не является, но имеет к нему отношение — *подстановка процесса*. Результат выполнения процесса можно представить в виде воображаемого файла и передать его другому процессу.

Форма вызова:

```
команда1 <(команда2)
```

При таком вызове процессу *команда1* передаётся файл (созданный налету канал или файл /dev/fd/...), в котором находятся данные, которые выводит *команда2*.

Например, у вас есть два файла с различными словами по одному в строке. Вы хотите определить, какие из них встречаются в одном файле, но не встречаются во втором. И наоборот.

```
%$ cat f1
```

```
a
b
c
b
b
%$ cat f2
a
c
c
c
%$ diff <(sort -u f1)  <(sort -u f2)
2d1
< b
```

Получается, что в первом файле присутствует слово, которое отсутствует во втором (слово *b*).

Подробнее см. Bash Process Substitution. 

8. Перенаправление вывода скрипта.

Потоки ввода/вывода скрипта, исполняющегося сейчас, изнутри самого скрипта можно следующим образом:

```
exec > file

exec 2>&1
```

Подробнее см. How do I redirect the output of an entire shell script within the script itself? 

9. Поменять местами стандартный поток вывода и стандартный поток ошибок.

```
$ cmd 3>&2 2>&1 1>&3
```

Подробнее см. IO Redirection - Swapping stdout and stderr (Advanced) 

10. Просмотр прогресса и скорости обработки данных в потоке.

Если пропустить данные, передающиеся в канале, через программу **pv**, то будет видна скорость обработки, время в течение которого работает канал и, если известно сколько данных должно быть обработано, приблизительное время окончания выполнения.

```
$ pv /tmp/shre.tar.gz | tar xz
```

62MB 0:00:18 [12,6MB/s] [==>   ] 10% ETA 0:02:27

Подробнее см. Pipe Viewer 1, Pipe Viewer 2.