Стандартные текстовые потоки и перенаправление

Одним из ключевых моментов в философии UNIX было то, что все команды CLI должны принимать текст как ввод и производить текст как вывод. Поскольку эта концепция применялась к разработке UNIX (и более поздних версий Linux), команды были разработаны для принятия текста в качестве ввода, выполнения некоторой операции над текстом и затем создания текста в качестве вывода. Команды, которые читают текст как ввод, изменяют этот текст некоторым образом и затем производят текст как вывод, иногда называют фильтрами.

Чтобы иметь возможность применять команды фильтра и работать с текстовыми потоками, полезно понимать несколько форм перенаправления, которые можно использовать с большинством команд: конвейеры, стандартное перенаправление вывода, перенаправление вывода ошибок и перенаправление ввода.

Стандартный выход

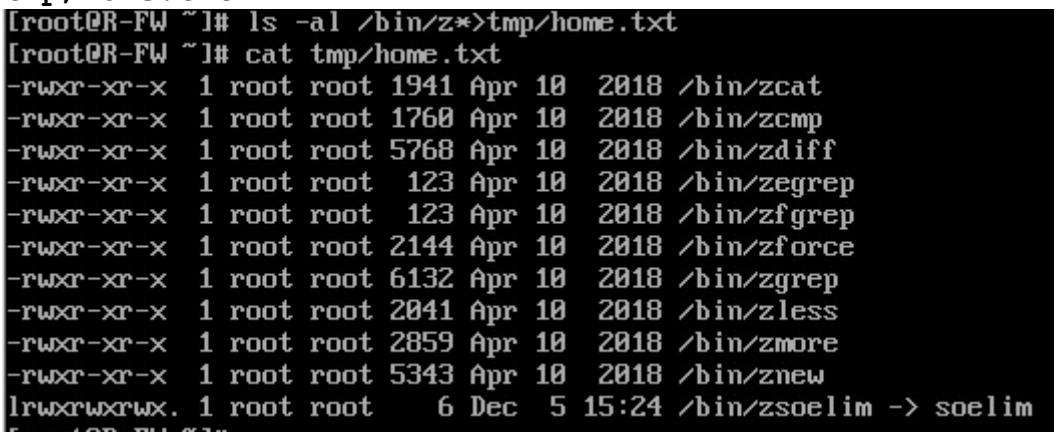
Когда команда выполняется без каких-либо ошибок, полученный вывод называется стандартным выходом (также называемым stdout или STDOUT). По умолчанию этот вывод будет отправлен на терминал, где выполняется команда. Можно перенаправить стандарт из команды, чтобы он перешел в файл вместо терминала.

Стандартное перенаправление вывода достигается с помощью команды с символом > (больше) и целевым файлом. Например, ls /etc/ выведет список файлов в домашнем каталоге. Чтобы сохранить список файлов в домашнем каталоге, вывод должен быть перенаправлен в текстовый файл. Создание файла /tmp/home.txt:

```
cd ~
mkdir tmp
ls /bin/z* > /tmp/home.txt
```

После чего home.txt будет выглядеть так:

```
cat /tmp/home.txt
```

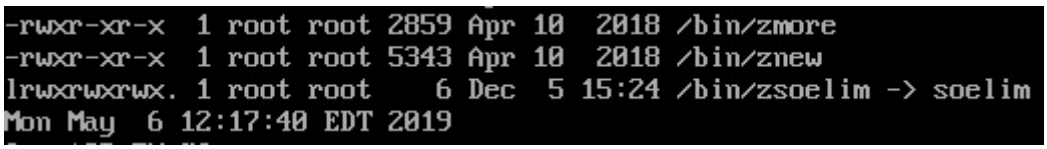


Перенаправление вывода с помощью одного > (символ больше) создаст новый файл или перезапишет содержимое существующего файла с тем же именем. Перенаправление стандартного вывода с двумя символами больше >>, также создаст новый файл, если он не существует. Разница в том, что вывод команды, использующей >>, будет добавлен в конец файла, если он уже существует. Например, чтобы добавить файл, созданный предыдущей командой, выполните следующее:

```
date >> /tmp/home.txt
```

Который будет обновлять home.txt до:

```
cat /tmp/home.txt
```



Существует номер, связанный со стандартным дескриптором выходного файла (символ >): номер один 1. Однако, поскольку стандартный вывод является наиболее часто перенаправленным выводом команды, этот номер можно опустить. Технически, команды должны быть выполнены, как показано ниже:

```
ls 1> /tmp/ls.txt
date 1>> /tmp/ls.txt
```

Стандартная ошибка

Когда команда встречает ошибку, она выдаст вывод, который известен как стандартная ошибка (также называемая **stderr** или **STDERR**). Как и **stdout**, вывод **stderr** обычно отправляется на тот же терминал, где в данный момент выполняется команда. Число, связанное со стандартным дескриптором файла ошибок, равно (два) **2.**

Если вы попытаетесь выполнить команду **ls /junk**, то команда выдаст стандартные сообщения об ошибках, поскольку каталог **/junk** не существует.

```
ls /junk
ls: cannot access /junk: No such file or directory
```

Поскольку эти выходные данные передаются в **stderr**, символ > не будет успешно перенаправлен, а выходные данные команды будут по-прежнему отправляться на терминал:

```
root@lin:~$ ls /junk > output
ls: cannot access /junk: No such file or directory
```

Чтобы перенаправить эти сообщения об ошибках, необходимо использовать правильный дескриптор файла, который для стандартной ошибки - номер 2. Выполните следующее, и ошибка будет перенаправлена в **/tmp/ls.err**:

```
ls /junk 2> /tmp/ls.err
```

Как и в случае стандартного вывода, использование одиночного > для перенаправления либо создаст файл, если он не существует, либо «перезаписает» (перезапишет) содержимое существующего файла. Чтобы предотвратить залипание существующего файла при перенаправлении стандартной ошибки, используйте двойной символ> после числа 2, чтобы добавить:

```
ls /junk 2>> /tmp/ls.err
```

Некоторые команды выдают как **stdout,** так и **stderr.** Эти два разных вывода могут быть перенаправлены в два отдельных файла с использованием следующего синтаксиса:

```
find /etc -name passwd > /tmp/output.txt 2> /tmp/error.txt
```

Иногда бесполезно отображать сообщения об ошибках в терминале или сохранять их в файле. Чтобы отменить эти сообщения об ошибках, используйте файл **/dev/null**.

Файл **/dev/null** похож на мусорное ведро, в котором все, что ему отправлено, исчезает из системы; его иногда называют «битым ведром» или «черной дырой». Любой тип вывода может быть перенаправлен в файл **/dev/null**; Чаще всего пользователи перенаправляют в этот файл стандартную ошибку, а не стандартный вывод.

Синтаксис для использования файла **/dev/null** такой же, как и для перенаправления в обычный файл:

```
find /etc -name passwd 2> /dev/null
```

Что если вы хотите, чтобы весь вывод (стандартная ошибка и стандартный вывод) отправлялся в один файл? Существует два способа перенаправления как стандартного вывода, так и стандартной ошибки:

```
root@lin:~$ ls > /tmp/ls.all 2>&1
root@lin:~$ ls &> /tmp/ls.all
```

Обе предыдущие командные строки создадут файл **/tmp/ls.all**, который будет содержать все **stdout** и **stderr**. Первая команда перенаправляет stdout в **/tmp/ls.all**, а **2> & 1** означает «отправлять **stderr** туда же, куда идет **stdout**». Во втором примере **&>** означает «перенаправить весь вывод».

Аналогичный метод может быть использован для добавления всех выходных данных в один файл:

```
root@lin:~$ ls /etc/au* >> /tmp/ls.all 2>&1
root@lin:~$ ls /etc/au* &>> /tmp/ls.all
```

Стандартный ввод

Стандартный ввод (также называемый **stdin** или **STDIN**) обычно осуществляется с клавиатуры с помощью ввода, предоставленного пользователем, который запускает команду. Хотя большинство команд могут читать входные данные из файлов, есть некоторые, которые ожидают, что пользователь введет его с помощью клавиатуры. Иногда полезно перенаправить стандартный ввод, чтобы он исходил из файла, а не с клавиатуры.

Хороший пример того, когда перенаправление ввода желательно, включает команду **tr**. Команда **tr** переводит символы, читая данные из стандартного ввода, переводя один набор символов в другой набор символов и затем записывая измененный текст в стандартный вывод.

Например, следующая команда **tr** будет принимать ввод от пользователя (через клавиатуру), чтобы выполнить перевод всех строчных букв в прописные. Выполните следующую команду, введите текст и нажмите ввод, чтобы увидеть перевод:

```
tr 'a-z' 'A-Z'
hello
HELLO
```

Это может привести к разочарованию после некоторого набора текста, потому что команда **tr** не прекращает чтение со стандартного ввода, пока не завершится или не получит символ «Конец передачи». Это можно сделать, набрав CTRL + D.

Команда **tr** не примет имя файла в командной строке в качестве аргумента. Чтобы выполнить перевод, используя файл в качестве входных данных, используйте перенаправление ввода. Чтобы использовать перенаправление ввода, введите команду с ее параметрами и аргументами, за которыми следует символ <(меньше символа) и путь к файлу, который будет использоваться для ввода. Например, создайте файл:

```
vi tmp/animals.txt
1 retriever
2 badger
3 bat
4 wolf
5 eagle

tr 'a-z' 'A-Z' < test/animals.txt
1 RETRIEVER
2 BADGER
3 BAT
4 WOLF
5 EAGLE
```

Выходные данные показывают файл animals.txt, «переведенный» на все заглавные буквы.

Предупреждение! Не пытайтесь использовать один и тот же файл для перенаправления ввода и вывода, поскольку результаты, вероятно, нежелательны (в итоге

вы теряете все данные). Вместо этого перехватите вывод и поместите его в другой файл, используйте другое имя файла, как показано ниже:

```
tr 'a-z' 'A-Z' < test/animals.txt > animals.new
```

Командные конвейеры

Чтобы эффективно использовать команды фильтра, часто используются конвейеры команд. В конвейере команд выходные данные одной команды отправляются другой команде в качестве входных данных. В Linux и большинстве операционных систем | (символ канала или вертикальная черта) используется между двумя командами для представления командного конвейера.

Например, представьте, что вывод команды history очень велик. Чтобы отправить этот вывод команде more, которая отображает одну «страницу» данных за раз:

```
history | more
```

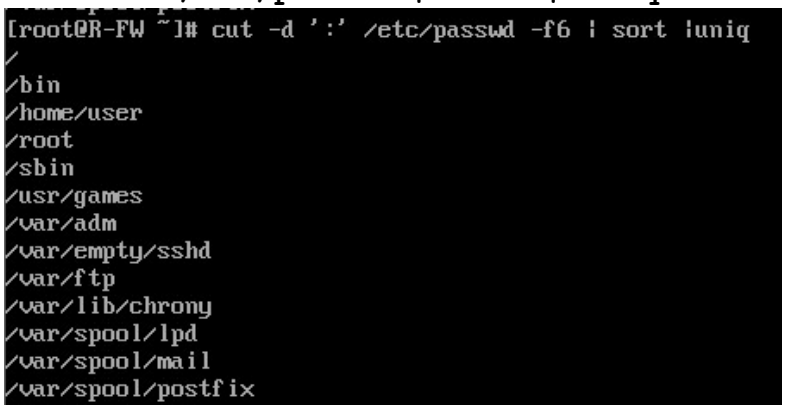
Более того, возьмите выходные данные команды **history** и отфильтруйте выходные данные с помощью команды **grep**:

```
history | grep "ls"
   11  ls ~ > /tmp/home.txt
   15  ls /junk
   16  ls /junk > output
   17  ls
   18  ls /junk 2> /tmp/au.err
   19  ls /junk 2>> /tmp/au.err
   23  ls
   25  ls -l error.txt
   29  ls /junk
   30  ls /junk > output
   32  ls /test
   33  ls test
   36  cat test/animals.txt
   37  tr 'a-z' 'A-Z' < test/animals.txt
   39  history | grep ls
   40  history | grep "ls"
```

В предыдущем примере текстовый вывод команды **history** перенаправляется в команду **grep** в качестве ввода. Команда **grep** сопоставляет строки **ls** и отправляет свой вывод в **stdout.**

Командные конвейеры становятся действительно мощными, когда объединяются три или более команд. Например, следующая командная строка извлекает некоторые поля из файла с помощью команды cut, затем сортирует эти строки с помощью команды sort и, наконец, удаляет повторяющиеся строки с помощью команды **uniq**:

```
cut -f6 -d':' /etc/passwd | sort | uniq
```



Администратор сервера может перенаправить вывод в разные места. Команда разбивает выходные данные команды на два потока: один направляется на терминал, а другой - в файл.

Для создания журнала команды или скрипта tee может быть очень полезным. Например, чтобы записать время выполнения процесса, начните с команды date и сделайте копию вывода в timer.txt:

```
date | tee timer.txt
Fri Nov  7 02:21:24 UTC 2014
```

Файл timer.txt теперь содержит копию даты:

```
cat timer.txt
Fri Nov  7 02:21:24 UTC 2014
```

Запустите процесс, который должен быть синхронизирован, сон 15 заменен синхронизированным процессом:

```
sleep 15
```

Затем снова введите команду date. На этот раз добавьте время к концу файла timer.txt с помощью параметра -a:

```
date | tee -a timer.txt
Fri Nov  7 02:28:43 UTC 2014
```

Оба раза будут записаны в файл.

```
root@lin:~$ cat timer.txt
Fri Nov  7 02:21:24 UTC 2014
Fri Nov  7 02:28:43 UTC 2014
```

Чтобы выполнить все вышеперечисленное как одну команду, используйте точку с запятой в качестве разделителя:

```
date | tee timer.txt; sleep 15; date | tee -a timer.txt
Fri Nov  7 02:35:47 UTC 2014
Fri Nov  7 02:36:02 UTC 2014
```

Файл timer.txt теперь содержит постоянный журнал времени выполнения.

```
cat timer.txt
Fri Nov  7 02:35:47 UTC 2014
Fri Nov  7 02:36:02 UTC 2014
```

Xargs Command

Команда **xargs** может использоваться для выполнения команды над многими файлами или несколько раз. Команда xargs может быть вызвана напрямую и примет любой ввод:

```
xargs
Hello
There
```

Действие по умолчанию команды xargs состоит в выводе вывода, если команда явно не следует за ним. После нажатия Ctrl + D команда xargs отправит ввод в команду echo:

```
echo '1a 1b 1c 1d' | xargs -n 2 touch
ls
1a  1c  Desktop    Downloads  Pictures  Templates  test
1b  1d  Documents  Music      Public    Videos
```

Эти четыре файла можно легко удалить, изменив команду touch на rm:

```
echo '1a 1b 1c 1d' | xargs -n 2 rm
ls
Desktop    Documents    Downloads    Music    Pictures    Public
Templates  Videos  test
```

Разделитель может быть установлен с помощью опции -d. Чтобы просмотреть содержимое каталога ~/test, содержащего слово alpha, со всеми экземплярами тире, замененными пробелом, введите следующее:

```
ls | grep alpha | xargs -d '-'
alpha first.txt
alpha first.txt.original
alpha second.txt
alpha third.txt
alpha.txt
```

Двойной канал в качестве вывода команды ls стал вводом команды grep и, наконец, команды xargs.

В следующем примере файл create-these-files.txt будет использоваться в качестве входных данных команды xargs:

```
cat ./create-these-files.txt
File1.txt
File2.txt
File3.txt
File4.txt
```

Перед запуском команды xargs содержимое каталога ~ /test:

```
root@lin:~/test$ ls
```

```
adjectives.txt            alpha.txt              letters.txt   os.csv
alpha-first.txt           animals.txt            linux.txt     people.csv
alpha-first.txt.original  create-these-files.txt  longfile.txt  profile.txt
alpha-second.txt          food.txt               newhome.txt   red.txt
alpha-third.txt           hidden.txt             numbers.txt
```

После запуска команды xargs с помощью touch появятся четыре новых файла:

```
cat ./create-these-files.txt | xargs touch
ls
```

```
File1.txt        alpha-first.txt.original  food.txt      numbers.txt
File2.txt        alpha-second.txt          hidden.txt    os.csv
File3.txt        alpha-third.txt           letters.txt   people.csv
File4.txt        alpha.txt                 linux.txt     profile.txt
adjectives.txt   animals.txt               longfile.txt  red.txt
alpha-first.txt  create-these-files.txt    newhome.txt
```

Команда xargs имеет множество других опций, которые можно изучить с помощью соответствующей страницы man.

Лабораторная работа «Файлы и потоки»

Текстовые файлы и потоки

1. Получите список всех процессов и перенаправьте вывод в файл ps.txt
2. Выполните команду поиска всех обычных файлов в каталоге /etc так, чтобы найденные имена файлов были записаны в файл SysComm в домашнем каталоге, а поток ошибок – в нуль-устройство /dev/null.
3. Выполните ту же команду, но так, чтобы потоки вывода и ошибок были записаны в файл SysComm.
4. Допишите в конец файла SysComm информацию о текущей дате и времени, выводимую командой date.
5. Установите опцию noclobber и сотрите содержимое файла ps.txt с помощью перенаправления вывода. Снимите опцию noclobber.
6. Проведите нерекурсивный поиск файлов символических ссылок в каталоге /usr/share/doc так, чтобы их список был выведен в отсортированном виде с помощью фильтра sort.
7. Выведите список всех процессов в файл ps.txt и отсортированный список на экран одновременно.
8. Выведите содержимое файла /etc/passwd в обратном порядке следования строк с нумерацией.
9. Получите имена трех самых больших файлов в каталоге /usr/bin
10. Получите столбец относительных приоритетов всех процессов в системе.
11. Получите столбец из имен пользователей, находящийся в данный момент в сеансе.
12. Получите список всех пользователей системы. В каком файле он находится?

13. С помощью sed получите список только тех процессов, которые связаны с каким-либо терминалом (фильтр по строке tty).
14. В полученном списке процессов с помощью sed замените все строки tty на terminal.