Изучаем Linux, 101: Потоки, программные каналы и перенаправления

Изучение основ работы с конвейерами Linux

Ян Шилдс (ishields@us.ibm.com)
Старший программист
IBM

29.11.2011

Если вы думаете, что потоки и каналы в Linux® каким-то образом связаны с водопроводом, то у вас есть отличный шанс разобраться в этом и узнать, как можно перенаправлять и разделять эти потоки. Вы также узнаете о том, как превратить поток в аргументы той или иной команды. Вы можете использовать этот материал для подготовки к экзамену LPI® 101 программы сертификации на администратора Linux начального уровня или просто для общего развития.

Больше статей из этой серии

Развить навыки по этой теме

Этот материал — часть knowledge path для развития ваших навыков. Смотри Основы системного администрирования Linux: работа с консолью

Краткий обзор

Из этой статьи вы узнаете об основных приемах перенаправления стандартных потоков ввода/вывода в Linux. Вы научитесь:

- Перенаправлять стандартные потоки ввода/вывода: стандартный поток ввода, стандартный поток вывода и стандартный поток ошибок.
- Направлять вывод одной команды на вход другой команды.
- Отправлять вывод одновременно на стандартное устройство вывода (stdout) и в файл.
- Использовать вывод команды в качестве аргументов другой команды.

Эта статья поможет вам подготовиться к сдаче экзамена LPI 101 на администратора начального уровня (LPIC-1) и содержит материалы цели 103.4 темы 103. Цель имеет вес 4.

Об этой серии

Эта серия статей поможет вам освоить задачи администрирования операционной системы Linux. Вы также можете использовать материал этих статей для подготовки к экзаменам первого уровня сертификации профессионального института Linux (LPIC-1).

Чтобы посмотреть описания статей этой серии и получить ссылки на них, обратитесь к нашему перечню материалов для подготовки к экзаменам LPIC-1. Этот перечень постоянно дополняется новыми статьями по мере их готовности и содержит самые последние (по состоянию на апрель 2009 года) цели экзаменов сертификации LPIC-1. Если какая-либо статья отсутствует в перечне, можно найти ее более раннюю версию, соответствующую предыдущим целям LPIC-1 (до апреля 2009 года), обратившись к нашим руководствам для подготовки к экзаменам института Linux Professional Institute.

Необходимые условия

Чтобы извлечь наибольшую пользу из наших статей, необходимо обладать базовыми знаниями о Linux и иметь работоспособный компьютер с Linux, на котором можно будет выполнять все встречающиеся команды. Иногда различные версии программ выводят результаты по-разному, поэтому содержимое листингов и рисунков может отличаться от того, что вы увидите на вашем компьютере.

Подготовка к выполнению примеров

Как связаться с Яном

Ян – один из наших наиболее популярных и плодовитых авторов. Ознакомьтесь со всеми статьями Яна (EN), опубликованными на сайте developerWorks. Вы можете найти контактные данные в профиле Яна и связаться с ним, а также с другими авторами и участниками ресурса My developerWorks.

Для выполнения примеров в этой статье мы будем использовать некоторые файлы, созданные ранее в статье "Изучаем Linux, 101: текстовые потоки и фильтры". Если вы не читали эту статью или не сохранили эти файлы, не расстраивайтесь! Давайте начнем с создания новой директории lpi103-4 и всех необходимых файлов. Для этого откройте текстовое окно и перейдите в вашу домашнюю директорию. Скопируйте содержимое листинга 1 в текстовое окно; в результате выполнения команд в вашей домашней директории будет создана поддиректория lpi103-4 и все необходимые файлы в ней, которые мы и будем использовать в наших примерах.

Листинг 1. Создание файлов, необходимых для примеров этой статьи

```
mkdir -p lpi103-4 && cd lpi103-4 && {
echo -e "1 apple\n2 pear\n3 banana" > text1
echo -e "9\tplum\n3\tbanana\n10\tapple" > text2
echo "This is a sentence. " !#:* !#:1->text3
split -l 2 text1
split -b 17 text2 y; }
```

Ваше окно должно выглядеть так, как показано в листинге 2, а вашей текущей рабочей директорией должна стать вновь созданная директория lpi103-4.

Листинг 2. Результаты создания необходимых файлов

```
[ian@echidna ~]$ mkdir -p lpi103-4 && cd lpi103-4 && {
> echo -e "1 apple\n2 pear\n3 banana" > text1
> echo -e "9\tplum\n3\tbanana\n10\tapple" > text2
> echo "This is a sentence. " !#:* !#:1->text3echo "This is a sentence. " "This is a sentence. " "This is a
 sentence.">text3
> split -l 2 text1
> split -b 17 text2 y; }
[ian@echidna lpi103-4]$
```

Перенаправление стандартного ввода/вывода

Командный интерпретатор Linux, такой как Bash, получает входные данные и направляет выходные данные в виде последовательностей или *потоков* символов. Любой символ не зависит ни от предыдущих, ни от последующих символов. Символы не упорядочены в виде структурированных записей или блоков с фиксированным размером. Доступ к потокам осуществляется с помощью механизмов ввода/вывода независимо от того, откуда поступают и куда передаются потоки символов (файл, клавиатура, окно, экран или другое устройство ввода/вывода). Командные интерпретаторы Linux используют три стандартных потока ввода/вывода, каждому из которых назначен определенный файловый дескриптор.

1. *stdout* – *стандартный поток вывода*, отображает вывод команд и имеет дескриптор 1.
2. *stderr* – *стандартный поток ошибок*, отображает ошибки команд и имеет дескриптор 2.
3. *stdin* – *стандартный поток ввода*, передает входные данные командам и имеет дескриптор 0.

Потоки ввода обеспечивают передачу входных данных (обычно поступающих с клавиатуры) командам. Потоки вывода обеспечивают печать текстовых символов, как правило, на терминал. Изначально терминал представлял собой ASCII печатающее устройство или дисплейный терминал, но сейчас, как правило, это просто окно на рабочем столе компьютера.

Если вы уже прочитали руководство "Изучаем Linux, 101: текстовые потоки и фильтры", то часть материала из этой статьи окажется вам знакомой.

Перенаправление вывода

Существует два способа перенаправления вывода в файл:

n>
: перенаправляет вывод из файлового дескриптора *n* в файл. Вы должны иметь права на запись в файл. Если файла не существует, то он будет создан. Если файл существует, то все его содержимое, как правило, уничтожается без каких-либо предупреждений.

n>>
: также перенаправляет вывод из файлового дескриптора *n* в файл. Вы также должны иметь права на запись в файл. Если файла не существует, то он будет создан. Если файл существует, то вывод добавляется к его содержимому.

Символ *n* в операторах n> или n>> является *файловым дескриптором*. Если он не указан, то предполагается, что используется стандартное устройство вывода. В листинге 3 продемонстрирована операция перенаправления для разделения стандартного потока вывода и стандартного потока ошибок команды `ls` с использованием файлов, которые были созданы ранее в директории lpi103-4. Также продемонстрировано добавление вывода команды в существующие файлы.

Листинг 3. Перенаправление вывода

```
[ian@echidna lpi103-4]$ ls x* z*
ls: cannot access z*: No such file or directory
xaa  xab
[ian@echidna lpi103-4]$ ls x* z* >stdout.txt 2>stderr.txt
[ian@echidna lpi103-4]$ ls w* y*
ls: cannot access w*: No such file or directory
yaa  yab
[ian@echidna lpi103-4]$ ls w* y* >>stdout.txt 2>>stderr.txt
[ian@echidna lpi103-4]$ cat stdout.txt
xaa
xab
yaa
yab
[ian@echidna lpi103-4]$ cat stderr.txt
ls: cannot access z*: No such file or directory
ls: cannot access w*: No such file or directory
```

Мы уже говорили, что перенаправление вывода с помощью оператора n> обычно приводит к перезаписи существующих файлов. Вы можете управлять этим свойством при помощи опции `noclobber` встроенной команды `set`. Если эта опция определена, то вы можете переопределить ее с помощью оператора n>|, как показано в листинге 4.

Листинг 4. Перенаправление вывода с помощью опции noclobber

```
[ian@echidna lpi103-4]$ set -o noclobber
[ian@echidna lpi103-4]$ ls x* z* >stdout.txt 2>stderr.txt
-bash: stdout.txt: cannot overwrite existing file
[ian@echidna lpi103-4]$ ls x* z* >|stdout.txt 2>|stderr.txt
[ian@echidna lpi103-4]$ cat stdout.txt
xaa
xab
[ian@echidna lpi103-4]$ cat stderr.txt
ls: cannot access z*: No such file or directory
[ian@echidna lpi103-4]$ set +o noclobber #restore original noclobber setting
```

Иногда может потребоваться перенаправить в файл как стандартный вывод, так и стандартный поток ошибок. Часто это используется в автоматизированных процессах или фоновых заданиях для того, чтобы впоследствии можно было просмотреть результаты работы. Чтобы перенаправить стандартный вывод и стандартный поток ошибок в одно и то же место, используйте оператор &> или &>>. Альтернативный вариант – перенаправить файловый дескриптор *n* и затем файловый дескриптор *m* в одно и то же место с помощью конструкции m>&n или m>>&n. В этом случае важен порядок перенаправления потоков. Например, команда

`command 2>&1 >output.txt`

это не то же самое, что команда

`command >output.txt 2>&1`

В первом случае поток ошибок stderr перенаправляется в текущее месторасположение потока stdout, а затем поток stdout перенаправляется в файл output.txt; однако второе перенаправление затрагивает только stdout, но не stderr. Во втором случае поток stderr перенаправляется в текущее месторасположение потока stdout, то есть, в файл output.txt. Эти перенаправления проиллюстрированы в листинге 5. Обратите внимание на последнюю

команду, в которой стандартный вывод был перенаправлен после стандартного потока ошибок, и, как следствие, поток ошибок продолжает выводиться в окно терминала.

Листинг 5. Перенаправление двух потоков в один файл

```
[ian@echidna lpi103-4]$ ls x* z* &>output.txt
[ian@echidna lpi103-4]$ cat output.txt
ls: cannot access z*: No such file or directory
xaa
xab
[ian@echidna lpi103-4]$ ls x* z* >output.txt 2>&1
[ian@echidna lpi103-4]$ cat output.txt
ls: cannot access z*: No such file or directory
xaa
xab
[ian@echidna lpi103-4]$ ls x* z* 2>&1 >output.txt # stderr does not go to output.txt
ls: cannot access z*: No such file or directory
[ian@echidna lpi103-4]$ cat output.txt
xaa
xab
```

В других ситуациях вам может потребоваться полностью проигнорировать стандартный вывод или стандартный поток ошибок. Для этого следует перенаправить соответствующий поток в пустой файл /dev/null. В листинге 6 показано, как проигнорировать поток ошибок команды `ls` и как с помощью команды `cat` убедиться в том, что файл /dev/null на самом деле пуст.

Листинг 6. Игнорирование стандартного потока ошибок посредством использования /dev/null

```
[ian@echidna lpi103-4]$ ls x* z* 2>/dev/null
xaa  xab
[ian@echidna lpi103-4]$ cat /dev/null
```

Перенаправление ввода

Так же, как мы можем перенаправить потоки stdout и stderr, мы можем перенаправить поток stdin из файла с помощью оператора <. Если вы прочли руководство "Изучаем Linux, 101: текстовые потоки и фильтры", то должны помнить, что в разделе Команды sort и uniq была использована команда `tr` для замены пробелов в файле text1 на символы табуляции. В том примере мы использовали вывод команды `cat` чтобы создать стандартный поток ввода для команды `tr`. Теперь для преобразования пробелов в символы табуляции вместо бесполезного вызова команды `cat` мы можем использовать перенаправление ввода, как показано в листинге 7.

Листинг 7. Перенаправление ввода

```
[ian@echidna lpi103-4]$ tr ' ' '\t'<text1
1       apple
2       pear
3       banana
```

В командных интерпретаторах, в том числе и в bash, реализована концепция *here-document*, которая является одним из способов перенаправления ввода. В ней используется

конструкция << и какое-либо слово, например END, являющееся маркером, или сигнальной меткой, означающей конец ввода. Эта концепция продемонстрирована в листинге 8.

Листинг 8. Перенаправление ввода с использованием концепции here-document

```
[ian@echidna lpi103-4]$ sort -k2 <<END
> 1 apple
> 2 pear
> 3 banana
> END
1 apple
3 banana
2 pear
```

Но почему нельзя просто набрать команду `sort -k2`, ввести данные и нажать комбинацию **Ctrl-d**, означающую конец ввода? Разумеется, вы могли бы выполнить эту команду, но тогда вы не узнали бы о концепции here-document, которая очень часто используется в сценариях командной оболочки (в которых не существует другой возможности указать, какие именно строки должны восприниматься в качестве ввода). Поскольку для выравнивания текста и обеспечения удобства чтения в сценариях широко используются символы табуляции, существует другой прием использования концепции here-document. При использовании оператора <<- вместо оператора << начальные символы табуляции удаляются.

В листинге 9 мы использовали подстановку команд для создания символа табуляции, а затем создали небольшой сценарий командной оболочки, содержащий две команды `cat`, каждая из которых считывает данные из блока here-document. Заметьте, что мы использовали слово END в качестве сигнальной метки блока here-document, который мы считываем с терминала. Если бы мы использовали это же слово в нашем сценарии, то наш ввод закончился бы преждевременно. Поэтому вместо слова END мы используем в сценарии слово EOF. После того, как наш сценарий создан, мы используем команду `.` (точка) чтобы запустить его в контексте текущего командного интерпретатора.

Листинг 9. Перенаправление ввода с использованием концепции here-document

```
[ian@echidna lpi103-4]$ ht=$(echo -en "\t")
[ian@echidna lpi103-4]$ cat<<END>ex-here.sh
> cat <<-EOF
> apple
> EOF
> ${ht}cat <<-EOF
> ${ht}pear
> ${ht}EOF
> END
[ian@echidna lpi103-4]$ cat ex-here.sh
cat <<-EOF
apple
EOF
        cat <<-EOF
        pear
        EOF
[ian@echidna lpi103-4]$ . ex-here.sh
apple
pear
```

В следующих статях этой серии вы узнаете больше о подстановке команд и сценариях. Ссылки на все статьи этой серии вы можете найти в перечне материалов для подготовки к экзаменам LPIC-1.

Создание конвейеров

В статье "Изучаем Linux, 101: текстовые потоки и фильтры" мы рассказывали о *фильтрации* как о процессе, в котором входной поток текста определенным образом преобразовывается и передается в выходной поток. Чаще всего такая фильтрация осуществляется через создание *конвейера* команд, в котором вывод одной команды *перенаправляется* на вход следующей команды. Подобное использование конвейеров не ограничено только текстовыми потоками, хотя чаще всего они применяются именно для работы с текстом.

Передача stdout в stdin

Оператор | (конвейеризация) помещается между двумя командами для направления потока stdout первой команды в поток stdin второй команды. Можно составлять более длинные конвейеры, добавляя дополнительные команды и операторы конвейеризации. Любая из команд может иметь опции или аргументы. Аргументом многих команд может являться не имя файла, а знак дефиса (-), который означает, что входные данные следует принимать со стандартного устройства ввода, а не из файла. Для полной уверенности вы можете обращаться к man-страницам нужной вам команды. Построение длинных конвейеров из команд, каждая из которых имеет свой ограниченный функционал – это распространенный в Linux и UNIX® прием, используемый для решения поставленных задач. В нашем гипотетическом конвейере (листинг 10) команды `command2` и `command3` имеют параметры, а команда `command3` использует параметр `-`, означающий ввод данных с устройства stdin.

Листинг 10. Конвейеризация вывода через несколько команд

```
command1 | command2 paramater1 | command3 parameter1 - parameter2 | command4
```

Необходимо заметить, что эти конвейеры передают **только** поток stdout в поток stdin. Вы не можете использовать оператор 2| для передачи потока stderr, по крайней мере с помощью тех инструментов, о которых вам известно к этому моменту. Если поток stderr был перенаправлен в поток stdout, то оба потока будут переданы по конвейеру. В листинге 11 приведен пример, в котором сначала используется команда `ls` с четырьмя аргументами, содержащими групповые символы и расположенными не в алфавитном порядке, а затем с помощью конвейера сортируются сообщения об ошибках и обычный вывод.

Листинг 11. Конвейер из двух потоков вывода

```
[ian@echidna lpi103-4]$ ls y* x* z* u* q*
ls: cannot access z*: No such file or directory
ls: cannot access u*: No such file or directory
ls: cannot access q*: No such file or directory
xaa  xab  yaa  yab
[ian@echidna lpi103-4]$ ls y* x* z* u* q*  2>&1 |sort
ls: cannot access q*: No such file or directory
ls: cannot access u*: No such file or directory
ls: cannot access z*: No such file or directory
xaa
xab
yaa
yab
```

Одним из преимуществ конвейеров в Linux и UNIX является то, что, в отличие от некоторых других популярных операционных систем, в конвейерах Linux не используются никакие промежуточные файлы. Стандартный поток вывода первой команды **не** записывается в файл, который потом считывается второй командой. Из руководства "Изучаем Linux, 101: управление файлами и директориями" вы уже знаете, как можно одновременно заархивировать и сжать файл с помощью команды `tar`. Если вам придется работать в операционной системе UNIX, в которой команда `tar` не имеет опции `-z` (сжатие с использованием gzip) или `-j` (сжатие с использованием bzip2), то это не беда. Для этого вы можете использовать, например, такой конвейер:

```
bunzip2 -c somefile.tar.bz2 | tar -xvf -
```

Создание конвейера, первым элементом которого является файл, а не поток stdout

Конвейеры, которые мы рассматривали выше, начинались с некоторой команды, генерирующей вывод, который, в свою очередь, передавался через каждый этап конвейера. Но что делать, если мы захотим начать наш конвейер с уже существующего файла, содержащего нужные данные? Это очень легко осуществить, поскольку многие команды могут принимать данные как со стандартного устройства ввода, так и из файлов. Если же используемый вами фильтр требует ввода данных с устройства stdin, то можно использовать команду `cat` для копирования содержимого файла в поток stdout. Этот способ сработает. Однако более распространенным решением является перенаправление ввода для первой команды и передача ее вывода через все остальные этапы конвейера. Для перенаправления потока stdin вашей первой команды в файл, который вы хотите обрабатывать, просто используйте оператор <.

Использование вывода в качестве аргументов

Из предыдущего материала нашей статьи вы узнали, как можно получить вывод одной команды и использовать его в качестве входных данных другой команды. Предположим теперь, что вы хотите использовать вывод команды или содержимое файла в качестве аргументов (а не входных данных) другой команды. В этом случае конвейеры вам не помогут, однако существуют три распространенных способа сделать это:

1. Команда `xargs`.

2. Команда `find` с опцией `-exec`.
3. Подстановка команд.

О двух первых способах вы узнаете из продолжения этой статьи. Пример подстановки команд вы видели в листинге 9, в котором мы создали символ табуляции. Подстановку команд можно использовать в командной строке, но чаще она используется в сценариях; обо всем этом вы узнаете из следующих статей нашей серии. Ссылки на все статьи этой серии вы можете найти в в перечне материалов для подготовки к экзаменам LPIC-1.

Использование команды `xargs`

Команда `xargs` считывает данные со стандартного устройства ввода, а затем строит и выполняет команды, параметрами которых являются полученные входные данные. Если не указана никакая команда, то используется команда `echo`. В листинге 12 приведен простой пример использования нашего файла text1, содержащего три строки по два слова в каждой.

Листинг 12. Использование команды `xargs`

```
[ian@echidna lpi103-4]$ cat text1
1 apple
2 pear
3 banana
[ian@echidna lpi103-4]$ xargs<text1
1 apple 2 pear 3 banana
```

Почему же тогда вывод `xargs` содержит только одну строку? По умолчанию `xargs` разбивает входные данные, если встречает символы-разделители, и каждый полученный фрагмент становится отдельным параметром. Однако когда `xargs` строит команду, ей за один раз передается максимально возможное количество параметров. Это поведение можно изменить с помощью параметра `–n` или `--max-args`. В листинге 13 приведен пример использования обоих вариантов; также был выполнен явный вызов команды `echo` для использования с `xargs`.

Листинг 13. Использование команд`xargs` и `echo`

```
[ian@echidna lpi103-4]$ xargs<text1 echo "args >"
args > 1 apple 2 pear 3 banana
[ian@echidna lpi103-4]$ xargs --max-args 3 <text1 echo "args >"
args > 1 apple 2
args > pear 3 banana
[ian@echidna lpi103-4]$ xargs -n 1 <text1 echo "args >"
args > 1
args > apple
args > 2
args > pear
args > 3
args > banana
```

Если входные данные содержат пробелы, но при этом они заключены в одиночные или двойные кавычки (либо пробелы представлены в виде escape-последовательностей с использованием обратной косой черты), то `xargs` не будет разбивать их на отдельные части. Это показано в листинге 14.

Листинг 14. Использование команды `xargs` и кавычек

```
[ian@echidna lpi103-4]$ echo '"4 plum"' | cat text1 -
1 apple
2 pear
3 banana
"4 plum"
[ian@echidna lpi103-4]$ echo '"4 plum"' | cat text1 - | xargs -n 1
1
apple
2
pear
3
banana
4 plum
```

До сих пор все аргументы добавлялись в конец команды. Если вам необходимо, чтобы после них были добавлены другие дополнительные аргументы, то воспользуйтесь опцией `-I` для указания строки замещения. В том месте вызываемой через `xargs` команды, в котором используется строка замещения, вместо нее будет подставлен аргумент. При использовании такого подхода каждой команде передается только один аргумент. Однако аргумент будет создан из целой входной строки, а не из отдельного ее фрагмента. Также вы можете использовать опцию `-L` команды `xargs`, в результате чего в качестве аргумента будет использоваться вся строка целиком, а не отдельные ее фрагменты, разделенные пробелами. Использование опции `-I` неявно вызывает использование опции `-L 1`. В листинге 15 приведены примеры использования опций `-I` и `–L`.

Листинг 15. Использование команды `xargs` и строк ввода

```
[ian@echidna lpi103-4]$ xargs -I XYZ echo "START XYZ REPEAT XYZ END" <text1
START 1 apple REPEAT 1 apple END
START 2 pear REPEAT 2 pear END
START 3 banana REPEAT 3 banana END
[ian@echidna lpi103-4]$ xargs -IX echo "<X><X>" <text2
<9      plum><9 plum>
<3      banana><3       banana>
<10     apple><10       apple>
[ian@echidna lpi103-4]$ cat text1 text2 | xargs -L2
1 apple 2 pear
3 banana 9 plum
3 banana 10 apple
```

Хотя в наших примерах используются простые текстовые файлы, вы не будете часто использовать команду `xargs` для таких случаев. Как правило, вы будете иметь дело с большим списком файлов, полученных в результате выполнения таких команд, как `ls`, `find` или `grep`. В листинге 16 показан один из способов передачи через `xargs` списка содержимого директории такой команде, как, например, `grep`.

Листинг 16. Использование команды `xargs` и списка файлов

```
[ian@echidna lpi103-4]$ ls |xargs grep "1"
text1:1 apple
text2:10        apple
xaa:1 apple
yaa:1
```

Что произойдет в последнем примере, если одно или несколько имен файлов будут содержать пробелы? Если вы попытаетесь использовать команду так, как это было сделано в листинге 16, то вы получите ошибку. В реальной ситуации список файлов может быть получен не от команды `ls`, а, например, в результате выполнения пользовательского сценария или команды; а может быть, вы захотите обработать его на других этапах конвейера с целью дополнительной фильтрации. Поэтому мы не берем во внимание тот факт, что вы могли бы просто использовать команду `grep "1" *` вместо существующей логической структуры.

В случае с командой `ls` вы могли бы использовать опцию `--quoting-style` для того, чтобы имена файлов, содержащие пробелы, были заключены в скобки (или представлены в виде escape-последовательностей). Лучшим решением (когда это возможно) является использование опции `-0` команды `xargs`, в результате чего для разделения входных аргументов используются пустые символы (\0). Хотя команда `ls` не имеет опции, позволяющей использовать в качестве вывода имена файлов с завершающим нулем, многие команды умеют делать это.

В листинге 17 мы сначала скопируем файл text1 в "text 1", а затем приведем несколько примеров использования списка имен файлов, содержащих пробелы, с командой `xargs`. Эти примеры позволяют понять саму идею, поскольку полностью освоить работу с `xargs` может оказаться не так просто. В частности, последний пример преобразования символов новой строки в пустые символы не сработал бы в том случае, если некоторые имена файлов уже содержали бы символы новой строки. В следующем разделе этой статьи мы рассмотрим более надежное решение с применением команды `find` для генерации подходящего вывода, в котором в качестве разделителей используются пустые символы.

Листинг 17. Использование команды `xargs` и файлов, содержащих пробелы в именах

```
[ian@echidna lpi103-4]$ cp text1 "text 1"
[ian@echidna lpi103-4]$ ls *1 |xargs grep "1" # error
text1:1 apple
grep: text: No such file or directory
grep: 1: No such file or directory
[ian@echidna lpi103-4]$ ls --quoting-style escape *1
text1  text\ 1
[ian@echidna lpi103-4]$ ls --quoting-style shell *1
text1  'text 1'
[ian@echidna lpi103-4]$ ls --quoting-style shell *1 |xargs grep "1"
text1:1 apple
text 1:1 apple
[ian@echidna lpi103-4]$ # Illustrate -0 option of xargs
[ian@echidna lpi103-4]$ ls *1 | tr '\n' '\0' |xargs -0 grep "1"
text1:1 apple
text 1:1 apple
```

Команда `xargs` не может строить сколь угодно длинные команды. Так, в Linux до версии ядра 2.26.3 максимальная длина команды была ограничена. Если вы попытаетесь выполнить такую команду, как, например, `rm somepath/*`, а директория содержит множество файлов с длинными именами, то выполнение может завершиться ошибкой, сообщающей, что список аргументов слишком длинный. Если вы работаете с более старыми версиями Linux или

UNIX, в которых могут присутствовать такие ограничения, то будет полезно узнать, как можно использовать `xargs` таким образом, чтобы обойти их.

Вы можете использовать опцию `--show-limits` для просмотра ограничений, установленных по умолчанию для команды `xargs`, и опцию `-s` – для задания максимальной длины выводимых команд. Об остальных опциях вы можете узнать из man-страниц.

Использование команды `find` с опцией `-exec` или совместно с командой `xargs`

Из руководства "Изучаем Linux, 101: управление файлами и директориями" вы узнали о том, как использовать команду `find` для поиска файлов на основе их имен, времени модификации, размера и прочих характеристик. Обычно над найденными файлами необходимо выполнять определенные действия – удалять, копировать, переименовывать их и так далее. Сейчас мы рассмотрим опцию `-exec` команды `find`, работа которой похожа на работу команды `find` с последующей передачей вывода команде `xargs`.

Листинг 18. Использование команды `find` с опцией `-exec`

```
[ian@echidna lpi103-4]$ find text[12] -exec cat text3 {} \;
This is a sentence.  This is a sentence.  This is a sentence.
1 apple
2 pear
3 banana
This is a sentence.  This is a sentence.  This is a sentence.
9       plum
3       banana
10      apple
```

Сопоставив результаты листинга 18 с тем, что вам уже известно о `xargs`, можно обнаружить несколько различий.

1. Вы **должны** использовать в команде символы {} для указания места подстановки, в которое будет подставлено имя файла. Эти символы не добавляются автоматически в конце команды.
2. Вы должны завершить команду точкой с запятой, которая должна быть представлена в виде escape-последовательности (\;, ';' или ";").
3. Команда выполняется один раз для каждого входного файла.

Попробуйте самостоятельно выполнить команду `find text[12] |xargs cat text3`, чтобы увидеть различия.

Теперь давайте вернемся к случаю, когда имя файла содержит пробелы. В листинге 19 мы попытались использовать команду `find` с опцией `-exec` вместо команд `ls` и `xargs`.

Листинг 19. Использование команды `find` с опцией `-exec` и файлов, содержащих пробелы в именах

```
[ian@echidna lpi103-4]$ find . -name "*1" -exec grep "1" {} \;
1 apple
1 apple
```

Пока все хорошо. Однако не кажется ли вам, что чего-то здесь не хватает? В каких файлах присутствовали строки, найденные при помощи `grep`? Здесь не хватает имен файлов, поскольку `find` вызывает `grep` один раз для каждого файла, а `grep`, будучи умной командой, знает о том, что если ей было передано имя лишь одного файла, то нет необходимости сообщать вам о том, что это был за файл.

В этой ситуации мы могли бы воспользоваться командой `xargs`, однако мы уже знаем о проблеме с файлами, имена которых содержат пробелы. Также мы упоминали тот факт, что команда `find` может генерировать список имен с пустыми разделителями, благодаря опции `-print0`. Современные версии команды `find` могут разделяться не точкой с запятой, а знаком +, благодаря чему, за один вызов команды `find` можно передать максимально возможное число имен, так же, как и в случае использования `xargs`. Излишне говорить о том, что в этом случае вы можете использовать конструкцию {} только один раз, и что она должна являться последним параметром команды. В листинге 20 продемонстрированы оба этих метода.

Листинг 20. Использование команд `find`, `xargs` и файлов, содержащих пробелы в именах

```
[ian@echidna lpi103-4]$ find . -name "*1" -print0 |xargs -0 grep "1"
./text 1:1 apple
./text1:1 apple
[ian@echidna lpi103-4]$ find . -name "*1" -exec grep "1" {} +
./text 1:1 apple
./text1:1 apple
```

Оба этих метода являются рабочими и выбор какого-то одного из них часто обусловлен лишь личными предпочтениями пользователя. Помните о том, что передавая по конвейеру объекты с необработанными символами-разделителями и пробелами, вы можете столкнуться с проблемами; поэтому если вы передаете вывод команде `xargs`, то используйте опцию `-print0` команды `find`, а также опцию `-0` команды `xargs`, которая сообщает, что во входных данных используются пустые разделители. Другие команды, включая `tar`, также поддерживают опцию `-0` и работу с входными данными, содержащими пустые разделители, поэтому всегда следует использовать эту опцию для тех команд, которые ее поддерживают, если только вы уверены на все 100%, что входной список не создаст вам проблем.

Наш последний комментарий затрагивает работу со списком файлов. Хорошей идеей будет всегда тщательно проверять ваш список и команды, прежде чем выполнять пакетные операции (например, удаление или переименование множества файлов). Наличие актуальной резервной копии в нужный момент также может оказаться неоценимым.

Разделение вывода

В завершение этой статьи мы кратко рассмотрим еще одну команду. Иногда может возникнуть необходимость просматривать вывод на экране и одновременно сохранять его в файл. Для этого вы **могли бы** перенаправить вывод команды в файл в одном окне, а затем

с помощью `tail -fn1` отслеживать вывод в другом окне, однако проще всего использовать команду `tee`.

Команда `tee` используется в конвейере, а ее аргументом является имя файла (или имена нескольких файлов), в который будет передаваться стандартный вывод. Опция `-a` позволяет не замещать старое содержимое файла новым содержимым, а добавлять данные в конец файла. Как уже говорилось при рассмотрении конвейеризации, если вы хотите сохранить как стандартный вывод, так и поток ошибок, то необходимо перенаправлять поток stderr в поток stdout прежде, чем передавать данные на вход команде `tee`. В листинге 21 приведен пример использования команды `tee` для сохранения вывода в два файла, f1 и f2.

Листинг 21. Разделение потока stdout с помощью команды tee

```
[ian@echidna lpi103-4]$ ls text[1-3]|tee f1 f2
text1
text2
text3
[ian@echidna lpi103-4]$ cat f1
text1
text2
text3
[ian@echidna lpi103-4]$ cat f2
text1
text2
text3
```

Ресурсы

- Оригинал статьи: Learn Linux, 101: Streams, pipes, and redirects (EN).
- Используйте перечень материалов для подготовки к экзаменам LPIC-1 для поиска статей developerWorks, которые помогут вам подготовиться к сдаче экзаменов программы сертификации LPIC-1, основанной на целях по состоянию на апрель 2009 года.
- На Web-сайте программы сертификации LPIC (EN) вы найдете подробные цели, списки задач и примерные вопросы всех трех уровней сертификации на администратора Linux-систем профессионального института Linux. В частности, на этом сайте представлены цели экзаменов LPI 101 и LPI 102 по состоянию на апрель 2009 года. Всегда обращайтесь к Web-сайту программы сертификации LPIC, чтобы узнать последние цели.
- Просмотрите всю серию статей для подготовки к экзаменам института LPI (EN) на сайте developerWorks, основанных на предыдущих целях, определенных до апреля 2009 года, чтобы изучить основы администрирования Linux и подготовиться к экзаменам для получения сертификата администратора Linux.
- Web-сайт Linux Documentation Project (EN) содержит большое количество полезной документации, в особенности, HOWTO-руководств.
- Посмотрите все советы по Linux (EN) и руководства Linux (EN) на сайте developerWorks.

Об авторе

Ян Шилдс

No bio.