

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ

Раздел #1

ИНТЕРФЕЙС КОМАНДНОЙ СТРОКИ

Командный язык и скрипты Shell

Лабораторная работа #12

Скрипты и перенаправление ввода/вывода



РАЗРАБОТАЛ

СЕРГЕЙ СТАНКЕВИЧ (SERHEY STANKEWICH, MINSK, BELARUS)

ЛАБОРАТОРНАЯ РАБОТА #12

Скрипты и перенаправление ввода/вывода

Цель работы

Закрепить на практике основы работы со сценариями командной оболочки и перенаправление ввода/вывода.

Краткие теоретические сведения.

Создание скриптов

Скрипты, их также называют **сценариями** командной строки, написанные для командного интерпретатора – оболочки «Shell». Сценарии командной строки, это наборы тех же самых команд, которые можно вводить с клавиатуры, записаны в виде текста, в обыкновенном текстовом файле. Эти файлы могут быть связаны между собой, то есть ссылаться друг на друга, и объединённые некоей общей целью, выполнять общую задачу. При этом результаты работы команд могут представлять либо самостоятельную ценность, либо служить входными данными для других команд.

Сценарии — это **мощный способ автоматизации** часто выполняемых действий. Чтобы успешно создать и запустить сценарий командной оболочки, нам нужно:

1. Написать сценарий.
2. Сделать сценарий выполняемым.
3. Поместить сценарий в каталог, где командная оболочка сможет найти его.

Как устроены *sh-скрипты

Создайте пустой файл с использованием команды `touch`. В его первой строке нужно указать, какую именно оболочку мы собираемся использовать. Нас интересует `bash`, поэтому первая строка файла будет такой:

```
#!/bin/bash
```

В других строках этого файла символ решётки используется для обозначения комментариев, которые оболочка не обрабатывает. Однако, первая строка – это особый случай. Здесь решётка, за которой следует восклицательный знак (эту последовательность называют *shebang*, произносится как «ше-банг») определяет путь к `bash`, то есть указывают системе на то, что сценарий должен быть выполнен интерпретатором `bash`.

Команды оболочки отделяются знаком перевода строки, **комментарии** выделяют знаком решётки:

```
#!/bin/bash
# This is a comment
pwd
whoami
```

Тут, так же, как и в командной строке, можно записывать команды в одной строке, разделяя точкой с запятой (`pwd; whoami`). Однако, если писать команды на разных строках, файл легче читать. В любом случае оболочка их обработает.

Важной особенностью скрипта является разделение полей. Причина заключается в специальной переменной окружения, которая называется `IFS` (Internal Field Separator) и позволяет указывать разделители полей. По умолчанию оболочка `bash` считает разделителями полей следующие символы:

- Пробел
- Знак табуляции
- Знак перевода строки

Если `bash` встречает в данных любой из этих символов, он считает, что перед ним – следующее самостоятельное значение списка. Для того, чтобы решить проблему, можно временно изменить переменную среды `IFS`. Вот как это сделать в `bash`-скрипте, если исходить из предположения, что в качестве разделителя полей нужен только перевод строки:

```
IFS=$'\n'
```

После добавления этой команды в `bash`-скрипт, он будет работать как надо, игнорируя пробелы и знаки табуляции, считая разделителями полей лишь символы перевода строки. Разделителями могут быть и другие символы. Например так:

```
IFS=:
```

Сделать сценарий выполняемым.

Система не позволяет интерпретировать любой старый текстовый файл как программу, и небезосновательно! Поэтому, чтобы выполнить сценарий, файлу сценария нужно дать разрешения на выполнение. Сделаем сценарий исполняемым при помощи команды `chmod`:

```
[me@linuxbox ~]$ ls -l hello_world
-rw-r--r-- 1 me me 63 2012-03-07 10:10 hello_world
[me@linuxbox ~]$ chmod 755 hello_world
[me@linuxbox ~]$ ls -l hello_world
-rwxr-xr-x 1 me me 63 2012-03-07 10:10 hello_world
```

Существует два распространенных набора разрешений для сценариев: 755 — для сценариев, которые должны быть доступны для выполнения всем, и 700 — для сценариев, которые могут выполняться только владельцами. Обратите внимание, что сценарии необходимо сделать доступными для чтения, чтобы их можно было выполнить.

Запуск сценария

После установки разрешений попробуем запустить сценарий, для этого необходимо добавить *явный путь* перед его именем:

```
[me@linuxbox ~]$ ./hello_world
```

В противном случае мы получим следующее сообщение:

```
[me@linuxbox ~]$ hello_world
bash: hello_world: команда не найдена
```

Специальная переменная окружения `PATH` влияет на то, как система ищет выполняемые программы. Система просматривает каталоги по списку всякий раз, когда требуется найти исполняемую программу, если путь к ней не указан явно. Именно так система выполняет программу `/bin/ls`, если мы вводим `ls` в командной строке.

Каталог `/bin` — один из каталогов, которые система просматривает автоматически. Список каталогов хранится в переменной окружения `PATH`. Она содержит список каталогов, перечисленных через двоеточие. Увидеть, что содержится в `PATH`, можно с помощью команды:

```
[me@linuxbox ~]$ echo $PATH
/home/me/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

Если поместить сценарий в любой из этих каталогов, проблема будет решена. Обратите внимание на первый каталог в списке, `/home/me/bin`. В большинстве дистрибутивов Linux в переменную `PATH` включается каталог `bin` в домашнем каталоге пользователя, чтобы дать пользователям возможность выполнять собственные программы. То есть если создать каталог `bin` и поместить сценарий в него, его можно будет запускать как любые другие программы:

```
[me@linuxbox ~]$ mkdir bin
[me@linuxbox ~]$ mv hello_world bin
[me@linuxbox ~]$ hello_world
Hello World!
```

Если каталог отсутствует в переменной `PATH`, его легко туда добавить, включив следующую строку в файл `.bashrc`:

```
export PATH=~/.bin:$PATH
```

Это изменение будет действовать в каждом последующем сеансе работы с терминалом. Чтобы применить изменения в текущем сеансе, нужно заставить командную оболочку повторно прочитать файл `.bashrc`, например, так:

```
[me@linuxbox ~]$ . .bashrc
```

Команда «точка» (`.`) является синонимом **source**, встроенной команды, которая читает указанный файл и интерпретирует его как ввод с клавиатуры.

Выбор местоположения для сценариев

Каталог `~/bin` хорошо подходит для сценария, этот сценарий предназначен для *личного использования*. Сценарии, которые должны быть доступны *всем пользователям* в системе, лучше размещать в традиционном местоположении — в каталоге `/usr/local/bin`. Сценарии, предназначенные для использования *системным администратором*, часто помещаются в каталог `/usr/local/sbin`.

В большинстве случаев программное обеспечение, созданное в локальной системе, будь то сценарии или скомпилированные программы, следует помещать в иерархию каталогов `/usr/local`, а не `/bin` или `/usr/bin`. Последние два каталога, как определено стандартом иерархии файловой системы Linux (Linux Filesystem Hierarchy Standard), предназначены только для файлов, поставляемых разработчиками дистрибутива Linux.

Перенаправление ввода/вывода данных

Самый мощный инструмент командной строки — это перенаправление ввода/вывода. Стандартные потоки ввода и вывода в Linux являются одним из

наиболее распространенных средств для обмена информацией процессов, а перенаправление потоков данных (`>`, `>>` и `|`) это одна из самых популярных конструкций командного интерпретатора.

Потоки данных

Стандартный ввод при работе пользователя в терминале передается через клавиатуру. Стандартный вывод и стандартная ошибка отображаются на дисплее терминала пользователя в виде текста. Данные выводятся на экран и считываются с клавиатуры, так как стандартные потоки по умолчанию ассоциированы с терминалом пользователя.

Ввод и вывод распределяется между тремя стандартными потоками данных: `stdin`, `stdout`, `stderr`.

Потоки можно подключать к чему угодно: к текстовым файлам, программам и даже устройствам. В командном интерпретаторе такая операция называется перенаправлением.

Поток	Значение потока	Номер файлового дескриптора потока
<code>stdin</code>	стандартный ввод (клавиатура)	0
<code>stdout</code>	стандартный вывод (экран)	1
<code>stderr</code>	стандартная ошибка (вывод ошибок на экран)	2

Команды с символами `>` или `<` означают перезапись существующего содержимого файла: `>` — стандартный вывод; `<` — стандартный ввод; `2>` — стандартная ошибка.

Команды с символами `>>` или `<<` не перезаписывают существующее содержимое файла, а присоединяют данные к нему: `>>` — стандартный вывод; `<<` — стандартный ввод; `2>>` — стандартная ошибка.

Примеры перенаправления с использованием файлов:

- **`< file`** — использовать файл как источник данных для стандартного потока ввода.
- **`> file`** — направить стандартный поток вывода в файл. Если файл не существует, он будет создан, иначе перезаписан сверху.
- **`2> file`** — направить стандартный поток ошибок в файл. Если файл не существует, он будет создан иначе перезаписан сверху.

- **>>file** — направить стандартный поток вывода в файл. Если файл не существует, он будет создан, иначе данные будут дописаны к нему в конец.
- **2>>file** — направить стандартный поток ошибок в файл. Если файл не существует, он будет создан, если существует — данные будут дописаны к нему в конец.
- **&>file** или **>&file** — направить стандартный поток вывода и стандартный поток ошибок в файл. Другая форма записи: **>file 2>&1**.

По сути, все, что происходит в операционной системе так или иначе связано с файлами. В экосистеме Linux (UNIX-системы) поддерживается концепция: «**Все есть файл**». Это значит, что любая сущность в операционной системе представлена в виде файла. Например, любая программа, любое устройство, такие как клавиатура, мышь, монитор, графический псевдо-терминал, жесткий диск и все остальные, представлены в виде файла.

Файл является первой основополагающей концепцией операционной системы и системного программирования.

Стандартный ввод

Стандартный входной поток обычно переносит данные от пользователя к программе. Программы, которые предполагают стандартный ввод, обычно получают входные данные от устройства типа клавиатура. Стандартный ввод прекращается по достижении EOF (конец файла), который указывает на то, что данных для чтения больше нет. Реальное значение EOF является отрицательным числом, зависящим от системы (в основном -1), что гарантирует несовпадение с кодом символа.

Чтобы прервать работу терминала мы можем нажать сочетание клавиш Ctrl+D. Это означает, что работа терминала достигла конца файла, EOF. А точнее мы сами ее туда направили.

Стандартный вывод

Стандартный вывод записывает данные, сгенерированные программой. Когда стандартный выходной поток не перенаправляется в какой-либо файл, он выводит текст на дисплей терминала.

Для работы со стандартным выводом используются следующие команды (утилиты), так же представленные в виде файлов в директории `/bin`:

cat – объединяет файлы;

echo – выводит на экран любой аргумент;

Команда cat. Это одна из самых полезных утилит, которые обязательно стоит выучить. Являясь сокращением английского слова «**concatenate**» (конкатенация), она позволяет создавать, объединять, а также выводить содержимое файлов в командной строке или в другом файле. Например, команда объединяет три файла: file1, file2 и file3 в один файл bigfile:

```
cat file1 file2 file3 > bigfile
```

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

Команда echo выводит на экран любой аргумент, который передается ему в командной строке.

Стандартная ошибка

Стандартная ошибка записывает ошибки, возникающие в ходе исполнения программы. Как и в случае стандартного вывода, по умолчанию этот поток выводится на терминал дисплея.

Например, введем команду **ls** с каталогом % в качестве аргумента:

```
$ ls %
```

Но так как каталога % не существует, на дисплей терминала будет выведен следующий текст стандартной ошибки:

```
ls: cannot access %: No such file or directory
```

Каналы

Механизм перенаправления дает возможность выполнять не только одиночные команды, но также составлять из команд целые конвейеры. Эта возможность обеспечивается специальным средством межпроцессного взаимодействия, каналами.

Канал – это механизм взаимодействия процессов. По каналу производится обмен данными между процессами. **Процессом** в операционной системе и системном программировании является программа, запущенная на исполнение, атрибуты и данные которой загружены в оперативную память компьютера. При работе с командной оболочкой процессом является набранная и

выполняемая команда. Канал (также иногда называемый трубой, *англ. pipe*) представляет собой объект, который можно использовать для связывания двух процессов.

Каналы бывают двух типов:

- Терминальный, «безымянный» канал
- Именованный канал

Терминальный канал. Терминальный канал в Linux – «безымянный», потому что существует анонимно и только во время выполнения процесса, в системе представлен как псевдофайл.

«Умение» команд Linux читать данные со стандартного ввода и выводить результаты в стандартный вывод используется механизмом командной оболочки, который называется *конвейером*.

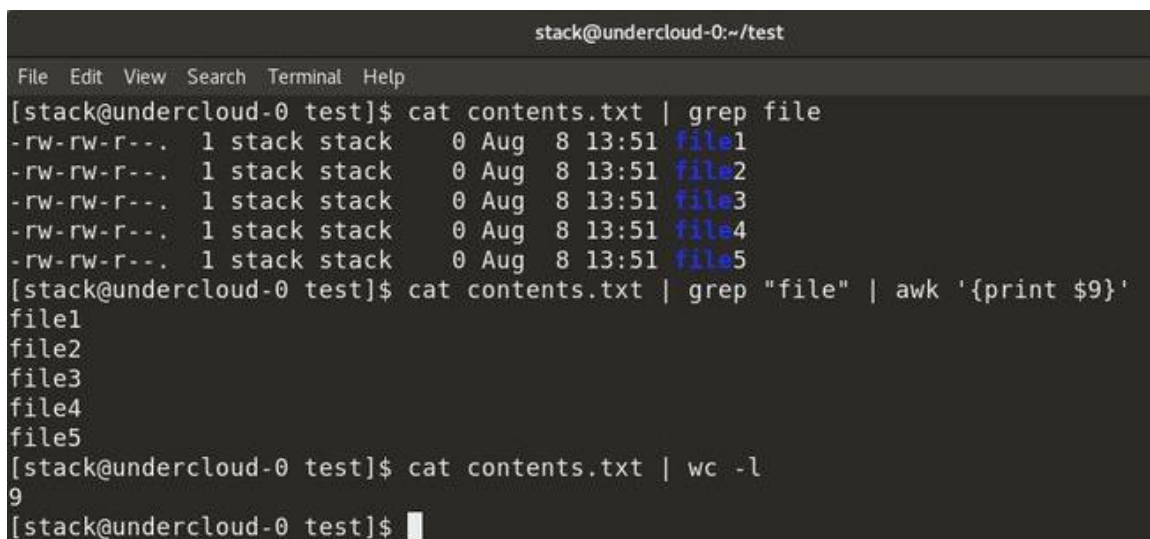
Конвейеры могут перенаправлять стандартный вывод, ввод или ошибку одного процесса другому для дальнейшей обработки.

Синтаксис системного вызова `pipe` или `pipe` без имени «спрятан» в символе (вертикальная черта) `|` между любыми двумя командами:

`Command-1 | Command-2 | ... | Command-N`

Поэтому часто этот оператор, `|`, называют также оператором **канала**.

Канал (конвейер) не может быть доступен в другом сеансе; он создается временно, чтобы обеспечить выполнение `Command-1` и перенаправить стандартный вывод. Он удаляется после успешного выполнения.



```

stack@undercloud-0:~/test
File Edit View Search Terminal Help
[stack@undercloud-0 test]$ cat contents.txt | grep file
-rw-rw-r--. 1 stack stack 0 Aug 8 13:51 file1
-rw-rw-r--. 1 stack stack 0 Aug 8 13:51 file2
-rw-rw-r--. 1 stack stack 0 Aug 8 13:51 file3
-rw-rw-r--. 1 stack stack 0 Aug 8 13:51 file4
-rw-rw-r--. 1 stack stack 0 Aug 8 13:51 file5
[stack@undercloud-0 test]$ cat contents.txt | grep "file" | awk '{print $9}'
file1
file2
file3
file4
file5
[stack@undercloud-0 test]$ cat contents.txt | wc -l
9
[stack@undercloud-0 test]$

```

Для демонстрации этого механизма нам понадобится несколько команд. Команда `less` может получать данные со стандартного ввода. Используем `less` для

постраничного отображения вывода любой команды, которая посылает свои результаты в стандартный вывод:

```
[me@linuxbox ~]$ ls -l /usr/bin | less
```

Данные первой программы, которые получает вторая программа, не будут отображаться. На дисплей терминала будут выведены только **отфильтрованные данные**, возвращаемые второй командой.

Это очень удобно! С помощью такого приема можно со всем комфортом исследовать вывод любой команды, посылающей результаты на стандартный вывод.

Именованные каналы. Именованный канал в UNIX-системах представлен в виде специального файла, имеющего тип – **pipe** (трубка, канал). Подробно именованные каналы мы изучим в курсе «Межпроцессное взаимодействие», в английском варианте это – Inter Process Communication (IPC).

Фильтры

Фильтры представляют собой стандартные команды Linux, которые могут быть использованы без каналов:

- wc** – выводит число символов перевода строки, слов и байтов в каждом указанном файле;
- grep** – находит и выводит строки, соответствующие шаблону;
- tee** – читает данные со стандартного ввода и записывает в стандартный вывод и в файлы;
- tr** – находит и заменяет одну строку другой.
- find** – возвращает файлы с именами, которые соответствуют передаваемому аргументу.

Как правило, все нижеприведенные команды работают как фильтры, если у них нет аргументов (опции могут быть):

- cut** – используется, если нужно вырезать часть текста;
- sort** – сортирует строки текста;
- uniq** – сообщает о повторяющихся строках или удаляет их;
- head** – выводит первые строки из файла;
- tail** – выводит последние строки из файла;

Команда cut. В Unix-системах эта команда удаляет секции текста, которые были обозначены при помощи байтов, символов или полей, разделенных знаками "-" и ":". Работу cut обеспечивает одноименная утилита. Если корректно использовать команду **cut** (вырезать) вместе с **sed**, **find** или **grep** в Linux/UNIX,

то можно получить много полезных отчетов о системе. Например, вы можете извлекать столбцы из файла.

Расширенный перечень команд представлен в разделе «Дополнительная информация».

Изучая мощный инструмент командной строки, перенаправление ввода/вывода, мы познакомились с главными основополагающими концепциями операционных систем и системного программирования: **файлами, процессами, каналами**. Более подробно мы изучим их на протяжении всего курса.

Best of LUCK with it, and remember to HAVE FUN while you're learning :)
Sergey Stankewich



УПРАЖНЕНИЯ

Упражнение 1

Наш первый скрипт

Теперь займемся написанием скриптов. Скрипт представляет последовательность команд Linux. Сценарии будем писать в текстовом редакторе. Для вызова текстового редактора набираем команду, например

```
$ gedit
```

Напишем простейший скрипт:

```
#!/bin/bash
read x
read y
let z="x+y"
echo $z
```

Скрипт всегда должен начинаться со строки **#!/bin/bash** или **#!/bin/sh**. Здесь **bash** и **sh** - это оболочки Linux, оболочка воспринимает команды с консоли и выполняет их. Заметим, что **bash** – более усовершенствованная версия, а версии **sh** в настоящее время вообще не существует.

В нашем скрипте команды **read x** (**read y**) читают значения переменных **x**, **y**. Команда **let z = "x+y"** выполняет сложение, а **echo \$z** выводит результат на экран.

Сохраните этот файл без всяких расширений, например, как **script_a**. Выполните его. Для этого наберите имя файла скрипта в текущей директории и нажмите клавишу **ENTER**.

Но тут могут возникнуть проблемы, скрипт не выполняется. Чтобы решить эту проблему скрипт нужно сделать **выполняемым** файлом. Этого мы добиваемся с помощью команды

```
$ chmod +x script_a
```

Теперь выполняем скрипт

```
$ ./script_a
```

Файлы, содержащие программы на языках сценариев, *дополнительно* должны быть *доступны для чтения*, иначе они не будут выполняться.

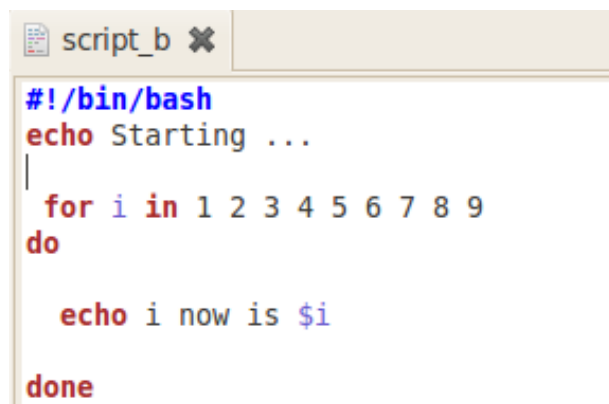
Однако если вы не изменили параметры доступа к файлу, не сделав его исполняемым, текст скрипта можно запустить, непосредственно указав название нужного командного интерпретатора и имя файла. Например,

```
$ dash script_a
```

Запустите команду **ls -l**, чтобы определить атрибуты прав доступа файла

```
[me@linuxbox ~]$ ls -l myscript1
-rw-rw-r-- 1 me me 0 2012-03-06 14:52 myscript1
```

Теперь введем цикл **for**. Синтаксис цикла **for** приведен на скриншотах ниже.



```
#!/bin/bash
echo Starting ...
for i in 1 2 3 4 5 6 7 8 9
do
    echo i now is $i
done
```

или

```
#!/bin/bash
for i in {1..20}
do
    echo i is now $i
    sleep 1
done
```

Здесь **\$i** – это значение переменной. Команда **sleep 1** осуществляет паузу в 1 секунду.

Цикл **while** представлен на примере ниже:

```
#!/bin/bash
i=100;
while [ $i -ge 0 ];
do
    echo counting down, from 100 to 0, now at $i;
    let i--;
done
```

Конструкция **if then else** позволяет выполнить действие, если выполняется условие, иначе – выполняется секция **else**.

```
#!/bin/bash
if [ -f isit.txt ]
then echo "file isit.txt exists!"
else
  echo "file isit.txt not found!"
fi
```

И выполнение самого скрипта, наберите его имя в текущей директории и нажмите клавишу ENTER:

```
ovgerman@ovgerman:~$ ./script18
file isit.txt not found!
```

Конструкция **if then elif**, если условий выбора несколько:

```
#!/bin/bash
count=42
if [ $count -eq 42 ]
then
  echo "42 is correct."
elif [ $count -gt 42 ]
then
  echo "Too much."
else
  echo "Not enough."
fi
```

Команда **find** позволяет найти имена файлов, заканчивающихся на “e1”

```
$ find -name “*e1”
```

Найти все файлы, заканчивающиеся на .conf:

```
$ find -type f -name “*.conf” > file2
```

Здесь результат выполнения операции записывается в файл с именем file2.

Найти все подкаталоги текущего каталога:

```
$ find -type d
```

Упражнение 2

Создадим песочницу для демонстрации работы конвейеров команд и фильтров

Создадим файл `basa.txt`, и запишем в него следующие строки:

```
Petrov: 20: 150: male  
Sidorov: 22: 50: male  
Nutrina: 21: 70: female  
Atomova: 19: 50: female
```

Мы видим, что записи разделены двоеточиями. Такие символы называют символами-разделителями (delimiters). Наш файл можно рассматривать как базу данных. Команда **cut** позволяет вывести столбцы, которые нас интересуют.

Например:

```
ovgerman@ovgerman:~$ cut -d":" -f1,3 basa.txt  
petrov:150  
sidorov:50  
nutrina:70  
utova:110  
ovgerman@ovgerman:~$
```

Здесь в команде `cut` указано, что делимитером (d) является знак двоеточия и что мы хотим вывести столбцы первый и третий (через запятую).

Следующий пример использует встроенный редактор строк (`sed`), который позволяет выполнить некоторые преобразования «на лету».

```
ovgerman@ovgerman:~$ cut -d":" -f1,3 basa.txt | sed 's/:/ /g'  
petrov 150  
sidorov 50  
nutrina 70  
utova 110  
ovgerman@ovgerman:~$
```

К предыдущей команде мы здесь добавили

| sed 's/:/ /g'

Первая буква `s` указывает, что будет производиться замена в строках. Далее указываем, что на что изменяет, а именно: заменяем на пустой символ. Буква `g` означает, что замена производится везде по всему тексту (global).

Команда `sed '/50/d' file2` удаляет строки, где содержится цифра 50.

```
Petrov:20:150:male
Sidorov:22:50:male
Sedina :21:70:female
ovgerman@ovgerman:~$ sed '/50/d' file2
Nutrina:21:70:female
ovgerman@ovgerman:~$
```

Команда **grep** ищет фрагмент текста, заданного образцом или выражением.

Например, пусть имеется файл, содержащий следующие строки:

```
$ cat tennis.txt
```

```
Amelie Mauresmo, Fra
Kim Clijsters, BEL
Justine Henin, Bel
Serena Williams, usa
Venus Williams, USA
```

```
$ cat tennis.txt | grep Williams
```

```
Serena Williams, usa
Venus Williams, USA
```

Здесь команда **grep** позволяет вывести людей по фамилии **Williams**.

Или

```
$ grep Williams tennis
```

```
Serena Williams, usa
Venus Williams, USA
```

Можно использовать **grep** без команды **cat**. Наконец, **grep \$x file1.txt** — выводит значение переменной **x** из файла.

Продолжим рассмотрение с команды **tr**. Эта команда заменяет какие-то буквы на другие. Вот пример замены маленькой буквы на большую.

```
ovgerman@ovgerman: ~
File Edit View Terminal Help
ovgerman@ovgerman:~$ echo hello world | tr 'w' 'W'
hello World
ovgerman@ovgerman:~$
```

Более сложный пример. Замена всех букв на большие.

```
ovgerman@ovgerman:~$ echo hello world | tr [a-z] [A-Z]
HELLO WORLD
ovgerman@ovgerman:~$
```


Параметры команды **tr**:

- c — использование всех символов, не перечисленных в аргументе *строка1*;
- d — удаление всех символов, перечисленных в аргументе *строка1*;
- s — замена нескольких повторяющихся символов на один символ.

Несколько примеров:

```
ovgerman@ovgerman:~$ cat file7
Privet year 2017
Good bue year 2016
ppprivet guys
ovgerman@ovgerman:~$ cat file7 | tr -s 'p ' >> temp7 | cat temp7
Privet year 2017
Good bue year 2016
privet guys
```

```
ovgerman@ovgerman:~$ cat file7
Privet year 2017
Good bue year 2016
ppprivet guys
ovgerman@ovgerman:~$ cat file7 | tr -c 'P' '+' > temp7
ovgerman@ovgerman:~$ cat temp7
P+++++++ovgerman@ovgerman:~$
```

Рассмотрим теперь команду сортировки **sort**. Создадим файл `basa2.txt` и зане-
сем в него значения

```
one
two
three
four
```

```
ovgerman@ovgerman:~$ echo one>basa2.txt
ovgerman@ovgerman:~$ echo two >> basa2.txt
ovgerman@ovgerman:~$ echo three>> basa2.txt
ovgerman@ovgerman:~$ echo four>>basa2.txt
ovgerman@ovgerman:~$ cat basa2.txt
one
two
three
four
ovgerman@ovgerman:~$ sort basa2.txt
four
one
three
two
ovgerman@ovgerman:~$
```

Команда `sort` сортирует именно по строкам, так что слова, расположенные в одной строке, не будут отсортированы.

Теперь поставим следующую задачу. Пусть дан файл `basa.txt` с содержимым, выведенным на скриншоте:

```
ovgerman@ovgerman:~$ cat basa.txt
Petrov:20:150:male
Sidorov:22:50:male
Nutrina:21:70:female
Atomova:19:50:female
ovgerman@ovgerman:~$
```

Требуется вывести фамилии в отсортированном виде. Решение реализовать в виде скрипта.

```
#!/bin/bash
cut -d":" -f1 temp.txt > temp2.txt
sort temp2.txt |>temp.txt
cat temp.txt
```

Можно сортировать записи по какому-то конкретному столбцу. Пример сортировки по второму столбцу (столбцы нумеруются с 1):

```
ovgerman@ovgerman:~$ sort -n -k2 t1.txt
Durik, 10, 0, male
Atomova, 19, 50, female
Petrov, 20, 150, male
Nutrina, 21, 70, female
Sidorov, 22, 50, male
ovgerman@ovgerman:~$ sort -n -k3 t1.txt
Durik, 10, 0, male
Atomova, 19, 50, female
Sidorov, 22, 50, male
Nutrina, 21, 70, female
Petrov, 20, 150, male
ovgerman@ovgerman:~$ sort -k4 t1.txt
Atomova, 19, 50, female
Nutrina, 21, 70, female
Durik, 10, 0, male
Petrov, 20, 150, male
Sidorov, 22, 50, male
ovgerman@ovgerman:~$ sed 's:/:/, /g' >t1.txt
```

```

ovgerman@ovgerman:~$ sed 's:/,/ /g' basa.txt>t1.txt
ovgerman@ovgerman:~$ cat t1.txt
Petrov, 20, 150, male
Sidorov, 22, 50, male
Nutra, 21, 70, female
Atomova, 19, 50, female
Durik, 10, 0, male
ovgerman@ovgerman:~$ sort -n -k2 t1.txt
Durik, 10, 0, male
Atomova, 19, 50, female
Petrov, 20, 150, male

```

Примечание. Предварительно символ ':' должен быть заменен на ',' (запятая с пробелом).

Параметры команды **sort**:

- n — означает числовую сортировку.
- k_n — означает сортировку по столбцу с номером n.

Команда comm (связь) позволяет показать, какие элементы есть у первого файла, которых нет у второго; какие элементы есть у второго файла, которых нет у первого, и какие элементы общие.

Создадим два текстовых файла: t1.txt , t2.txt. Заполните файлы так, чтобы у них были одинаковые и отличные, уникальные данные.

```

File Edit View Terminal Help
ovgerman@ovgerman:~$ touch t1.txt
ovgerman@ovgerman:~$ touch t2.txt
ovgerman@ovgerman:~$ echo one>>t1.txt
ovgerman@ovgerman:~$ echo two>>t1.txt
ovgerman@ovgerman:~$ echo four>>t1.txt
ovgerman@ovgerman:~$ echo seven>>t1.txt
ovgerman@ovgerman:~$ echo two>>t2.txt
ovgerman@ovgerman:~$ echo four>>t2.txt
ovgerman@ovgerman:~$ echo six>>t2.txt
ovgerman@ovgerman:~$ cat t1.txt
one
two
four
seven
ovgerman@ovgerman:~$ cat t2.txt
two
four
six
ovgerman@ovgerman:~$

```

Чтобы использовать команду **comm** надо сначала отсортировать содержимое файлов.

```

ovgerman@ovgerman:~$ sort t1.txt>t11.txt
ovgerman@ovgerman:~$ cat t11.txt
four
one
seven
two
ovgerman@ovgerman:~$ touch t22.txt
ovgerman@ovgerman:~$ sort t2.txt>>t22.txt
ovgerman@ovgerman:~$ cat t22.txt
four
six
two
ovgerman@ovgerman:~$

```

Теперь можно выполнить команду `comm`:

```

ovgerman@ovgerman:~$ comm t11.txt t22.txt
      four
one
seven
      six
      two
ovgerman@ovgerman:~$

```

Можно попробовать вывести не все три столбца, а только один или два. Для этого в команде нужно указать, какие столбцы исключить. Так, выведем столбец общих элементов

```

ovgerman@ovgerman:~$ comm -12 t11.txt t22.txt
four
two
ovgerman@ovgerman:~$

```

Здесь указано, что столбцы 1 и 2 выводить не надо. Будет выведен только третий столбец общих элементов.

Команда **head** выводит первые десять записей файла:

```

paul@debian7~$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh

```

Команда **head** используется также для отображения первых *n* записей:

```
paul@debian7~$ head -4 /etc/passwd
```

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
paul@debian7~$
```

Команда **tail** аналогична **head**, но отображает *n* записей с конца файла (если *n* не указано, то это последние десять записей).

Упражнение 3

Подключение кода из внешнего файла

В скриптах имеется средство, которое позволяет запускать другие скрипты и программы из внешних файлов. Это повышает гибкость программирования и позволяет делать скрипты *бесконечно сложными*.

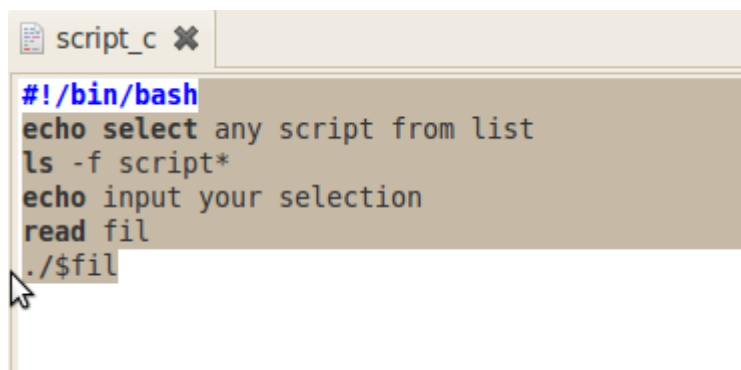
Для этого используется команда «**.**» или слово **source**, представляющая аналог *#include* (в языке Си).

```
source file # подключение кода Bash из файла
```

или

```
ls | sed ... | ./dev/stdin # подключение кода (полученного путём
выполнения предыдущей команды) из стандартного входа
```

Например, можно написать скрипт, в котором предлагается выбрать скрипт из списка и выполнить его.



```
#!/bin/bash
echo select any script from list
ls -f script*
echo input your selection
read fil
./$fil
```

We hope you enjoy working with Linux!



ЗАДАНИЯ

Задание 1

1. Написать скрипт, который записывает содержимое одного файла в другой и при этом оба файла существуют до операции. Дать два разных варианта выполнения.
2. Найти все файлы, начинающиеся на слово `script` и записать их имена в файл `list.txt`.
3. Написать скрипт, который проверяет, содержится ли скриптовый файл в директории, если да, то выполнить его.
4. Написать скрипт, подсчитывающий сумму от 1 до 10.
5. А какие командные интерпретаторы установлены в вашей системе? Напишите скрипт поиска местонахождения командных интерпретаторов вашей системы. Также определите какой интерпретатор используется сейчас в вашей системе?

Задание 2

6. Используйте текст первого примера, заменить некоторые фамилии на фамилии *членов вашей команды* (например, `Nutrina` заменить на `Stankewich`).
7. Отсортируйте данные файла так, чтобы он не поменял свое собственное имя. (Решение. Нужно предварительно создать промежуточный отсортированный файл `t11.txt`. Затем выполнить нужные команды).
8. Подсчитайте число одинаковых слов в обоих файлах. Для подсчета числа слов в файле используйте команду: **`wc -w file1`**.
9. Написать скрипт, который создает отсортированный файл, содержащий слова из двух файлов, исключая их общую часть одинаковых слов.
10. Вывести фамилию самого молодого человека (в файле `basa.txt` – это второй столбец).
11. Вывести зарплату самого молодого человека (зарплата – третий столбец).
12. Вывести отсортированный список имен файлов, начальная часть имени есть `script`.
13. Написать скрипт, который в каждой строке файла оставляет только буквенные символы, а остальные символы выбрасывает.

Пример.

Пусть содержимое файла есть:

Privet year 2022

Good bye year 2021.

Должны получить такой файл:

Privet year

Good bye year

14. Вывести упорядоченный список имен файлов, в именах которых содержится символ подчеркивания, например, `script_sort`.

Задание 3

15. Напишите скрипт запуска другого скрипта из текущей директории.
16. Создайте два скрипта с одинаковым именем, но разным текстом. Эти скрипты должны быть в разных директориях. В третьей директории создайте скрипт, запускающий два первых скрипта. При этом используйте команду **`sours`**. Объясните поведение программы и расскажите с какими проблемами вы столкнулись.
17. Расположите скрипт в одной из *стандартных* директорий системы, и запустите его из *домашней* директории пользователя.
18. Напишите скрипт запускающий два и более терминалов отдельно (количество терминалов зависит от количества голов в команде). Количество терминалов должно вводиться интерактивно. Желательно чтобы в этих терминалах запускались какие нибудь программки, можно запустить браузер.

«Easy things should be easy and hard things should be possible»
«Простые вещи должны быть простыми, а сложные вещи должны быть возможными»



Контрольные вопросы:

Скрипты или сценарии командной оболочки

- 1) Что такое скрипт или сценарий командной оболочки, для чего они используются?
- 2) Какими правами доступа должны быть наделены файлы скриптов?
- 3) Какие способы запуска выполнения скриптов вы знаете? Как запустить скрипт на который отсутствуют права исполнения (запуска)?
- 4) Какое значение имеет месторасположение скриптов в файловой системе для их запуска?
- 5) Какие стандартные каталоги системы Linux вы знаете, и для скрипты и программ каких пользователей эти каталоги предназначены?
- 6) Что такое переменная окружения PATH, и какую роль она играет в запуске скриптов?

Перенаправление ввода/вывода и потоки данных

- 7) Что такое перенаправление ввода/вывода и потоки данных?
- 8) Что такое потоки данных, и какие стандартные потоки данных существуют в системе? В какой системной директории располагаются их файлы?
- 9) Назовите три основополагающие концепции операционных систем и системного программирования?
- 10) Для чего используются команды-фильтры? Назовите некоторые из них.

Дополнительная информация

Подробно песочница представлена к книге: Шоттс У. «Командная строка Linux. Полное руководство.» — СПб.: Питер, 2017. — 480 с.: ил. — (Серия «Для профессионалов») на страницах [76, 477, 346-353](#).

Робачевский А. М. Операционная система UNIX®. - СПб.: 2002. - 528 ил.

Команды работают как фильтры, если у них нет аргументов (опции могут быть):

- **cat** — считывает данные со стандартного потока ввода и передает 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 передает на поток вывода. В качестве разделителей полей могут использоваться любые символы.
- **sort** — сортирует данные в соответствии с какими-либо критериями, например, арифметически по второму столбцу.
- **uniq** — удаляет 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.
- **bc** — вычисляет каждую отдельную строку потока и записывает вместо нее результат вычисления.
- **hexdump** — показывает шестнадцатеричное представление данных, поступающих на стандартный поток ввода.
- **strings** — выделяет и показывает в стандартном потоке (или файле) то, что напоминает строки. Всё что не похоже на строковые последовательности, игнорируется. Команда полезна в сочетании с **grep** для поиска интересных строковых последовательностей в бинарных файлах.
- **sed** — обрабатывает текст в соответствии с заданным скриптом. Наиболее часто используется для замены текста в потоке: **sed s/было/стало/g**.
- **awk** — обрабатывает текст в соответствии с заданным скриптом. Как правило, используется для обработки текстовых таблиц, например, вывод **ps aux** и т.д.

- **sh -s** — текст, который передается на стандартный поток ввода sh -s. может интерпретироваться как последовательность команд shell. На выход передается результат их исполнения.
- **ssh** — средство удаленного доступа ssh, может работать как фильтр, который подхватывает данные, переданные ему на стандартный поток ввода, затем передает их на удаленный хост и подает 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) передается со стандартного вывода ssh.

Если в качестве аргумента передается файл, команда-фильтр считывает данные из этого файла, а не со стандартного потока ввода (есть исключения, например, команда `tr`, обрабатывающая данные, поступающие исключительно через стандартный поток ввода).

Интернет источники

Перенаправление ввода/вывода в Linux

<https://selectel.ru/blog/tutorials/linux-redirection/>

http://tdkare.ru/sysadmin/index.php/Программирование_на_bash



Переменные окружения

