Лабораторная работа №3
«BASH: ПОТОКИ ДАННЫХ. ПРОГРАММИРОВАНИЕ»

Часть 1. Потоки ввода и вывода данных

Потоки и файлы

Логически все файлы в системе Linux организованы в непрерывный поток байтов. Любой файл можно свободно копировать и добавлять к другому файлу, так как все файлы организованы одинаково.

Эта логическая организация файлов распространяется на операции ввода и вывода. Данные в операциях ввода и вывода организованы аналогично файлам. Данные, вводимые с клавиатуры, направляются в поток данных, организованный как непрерывная совокупность байтов. Данные, выводимые из команды или программы, также направляются в поток, организованный как непрерывная совокупность байтов. Входной поток данных в ОС Linux называется стандартным вводом, а выходной поток данных - стандартным выводом.

Поскольку стандартный ввод и стандартный вывод организованы так же, как файл, они свободно могут взаимодействовать с файлами. В ОС Linux широко используется переадресация, которая позволяет перемещать данные в файлы и из файлов.

Переадресация стандартного вывода: > и >>

Когда выполняется команда ОС Linux, дающая какую-либо выходную информацию, эта информация направляется в поток данных стандартного вывода. По умолчанию в качестве пункта назначения данных стандартного вывода используется какое-либо устройство, в данном случае экран. Устройства, такие как клавиатура и экран, тоже рассматриваются как файлы. Они принимают и отправляют потоки байтов, имеющих такую же организацию, как и файлы байтового потока. Экран - это устройство, на котором отображается непрерывный поток байтов. По умолчанию стандартный вывод посылает свои данные на экран, где они отображаются.

Для направления стандартного вывода в файл, а не на экран, необходимо использовать оператор переадресации вывода: >. С помощью операции переадресации создается новый файл-адресат. Если он уже существует, то система заменит его содержимое данными стандартного вывода. Для того чтобы этого не произошло, можно установить для shell режим noclobber:

```
set -o noclobber
```

В этом случае операция переадресации существующего файла выполнена не будет. Отменить режим noclobber можно, поставив после оператора переадресации восклицательный знак:

```
cat file1 >! file2
```

Хотя оператор переадресации и имя файла ставятся после команды, перенаправление стандартного потока выполняется не после выполнения команды, а до него. С помощью оператора переадресации создается файл и переадресация организуется до того, как начинают поступать данные со стандартного вывода. Если файл уже существует, он будет разрушен и заменен новым файлом под тем же именем. Команда, генерирующая выходные данные, выполняется только после создания файла переадресации.

Если пользователь попытается использовать одно и то же имя для входного файла команды и переадресованного файла, возникнет ошибка. Так как операция переадресации выполняется первой, входной файл, поскольку он существует, разрушается и заменяется файлом с тем же именем. Когда команда начинает выполняться, она обнаруживает пустой входной файл. Для добавления стандартного вывода к существующему файлу служит оператор переадресации >>.

Переадресация стандартного ввода: < и <<

Многие команды ОС Linux могут принимать данные со стандартного ввода. Сам стандартный ввод получает данные из устройства или из файла. По умолчанию в качестве устройства для стандартного ввода используется клавиатура. Символы, набираемые на клавиатуре, подаются на стандартный ввод, который затем направляется в команду.

Во многих Linux-системах применяется метод буферизации строк. При буферизации строк, информация посылается на стандартный ввод только после того, как пользователь ввел всю строку.

Стандартный ввод можно переадресовать так же, как и стандартный вывод. Стандартный ввод может приниматься не с клавиатуры, а из файла. Оператор переадресации стандартного ввода: <. Кроме этого для переадресации стандартного ввода применяется механизм "файл здесь":

```
cat << 'слово-признак конца ввода'
> 'текст'
> 'текст'
> 'слово-признак конца ввода'
```

Операции переадресации стандартного ввода и стандартного вывода можно объединять.

Переадресация и пересылка по каналу стандартного потока ошибок: >&, 2>.

При выполнении команд иногда происходят ошибки. Например, пользователь указал неверное количество аргументов или возникла какая-то системная ошибка. Когда возникает ошибка, система выдает специальное сообщение. Как правило, такие сообщения об ошибках отображаются на экране вместе со стандартным выводом. ОС Linux, однако, различает стандартный вывод и сообщения об ошибках. Сообщения об ошибках помещаются еще в один стандартный байтовый поток, который называется стандартным потоком ошибок (диагностики).

Так как сообщения об ошибках направляются в поток, отдельный от стандартного вывода, то в случае переадресации стандартного вывода в файл они все равно появляются на экране.

Стандартный поток ошибок можно переадресовать так же, как и стандартный вывод. Например, сообщения об ошибках можно сохранить в файле для справок. Как и в случае стандартного вывода, пунктом назначения стандартного потока ошибок по умолчанию является экран, однако с помощью специальных операторов переадресации его можно переадресовать в любой файл или устройство.

Для переадресации стандартного потока ошибок в shell предусмотрена специальная возможность. Все стандартные байтовые потоки в операциях переадресации можно обозначать номерами (дескрипторами). Номера 0, 1 и 2 обозначают соответственно стандартный ввод, стандартный вывод и стандартный поток ошибок. Оператор переадресации вывода, >, по умолчанию действует на стандартный вывод, 1. Чтобы переадресовать стандартный поток ошибок, нужно поставить перед оператором переадресации вывода цифру 2.

Стандартный поток ошибок можно дописать в файл, используя цифру 2 и оператор добавления: >>. Для того чтобы переадресовать и стандартный вывод, и стандартный поток ошибок, нужны две операции переадресации и два файла.

В BASH можно ссылаться на стандартный поток по его номеру со знаком "&": &1 обозначает стандартный вывод. Это обозначение можно использовать в операции переадресации для того, чтобы сделать стандартный вывод файлом назначения. Операция переадресации 2>&1 переадресует стандартный поток ошибок на стандартный вывод. В результате стандартный вывод становится файлом назначения для стандартного потока ошибок. Операция переадресации 1>&2 переадресует стандартный ввод в стандартный поток ошибок. По умолчанию входным потоком операции >& является стандартный поток ошибок, а выходным потоком - стандартный вывод. Поэтому, если его использовать в команде, все сообщения о ошибках будут перенаправляться на стандартный вывод.

Программные каналы: |

Иногда возникают ситуации, когда нужно передать данные из одной команды в другую. Другими словами, необходимо послать стандартный вывод одной команды на стандартный ввод другой, а не в файл. Для образования такого соединения в ОС Linux используется так называемый канал. Оператор канала, | (вертикальная черта), помещенный между двумя командами, связывает их стандартные потоки. Стандартный вывод одной команды становится стандартным вводом другой. Выходная информация команды, стоящей перед оператором канала, передается в качестве входной в команду, стоящую за оператором канала.

Программные каналы можно объединять с другими средствами shell, например со специальными символами, проводя таким образом специализированные операции.

Если операция переадресации позволяет просто направлять выходную информацию в файл, то каналы обеспечивают ее пересылку в другую команду Linux. Следует помнить о различии между файлом и командой. Файл - это носитель данных. Вы можете хранить или читать из него данные. Команда - это программа, которая исполняет инструкции. Команда может читать данные из файла и сохранять данные в файле, но во время выполнения ее нельзя рассматривать как файл. По этой причине операция переадресации выполняется с файлами, а не с командами. В процессе переадресации данные посылаются из программы в файл, а не в другую

программу. Пунктом назначения операции переадресации могут быть только файлы, но не программы.

Можно, тем не менее, смоделировать процесс конвейерной пересылки с помощью нескольких операций переадресации. Выходная информация одной команды посылается в файл. Команда, записанная в следующей строке, использует этот файл как переадресованный ввод.

Каналы работают со стандартным выводом команды независимо от того, что подается на этот вывод. Пересылаться по каналу из одной команды в другую может содержимое целого файла и даже нескольких файлов.

Стандартным вводом, посылаемым по каналу в команду, можно более эффективно управлять с помощью аргумента стандартного ввода, `-`. Дефис, используемый в команде в качестве аргумента, обозначает стандартный ввод.

Допустим, нужно напечатать файл с именем его каталога в начале списка. Команда pwd выдает имя каталога, а команда cat выдает содержимое файла. В данном случае команде cat нужно использовать в качестве входной информации и файл, и стандартный ввод, пересланный по каналу из команды pwd. Команда cat будет иметь два аргумента: стандартный ввод, обозначенный дефисом, и имя выводимого на печать файла.

```
pwd | cat - file | lpr
```

Каналы и переадресация: команда tee

Для того, чтобы переадресовать стандартный вывод в файл и одновременно воспроизвести эту информацию на экране необходимо использовать команду tee. Команда tee копирует стандартный вывод в файл. В качестве аргумента она использует имя нового файла, в который копируется стандартный вывод. Это все равно что скопировать содержимое стандартного вывода и один его экземпляр переадресовать в файл, а другой отправить дальше (часто - на экран).

Переадресацией в сочетании с каналами необходимо пользоваться осторожно. Переадресация стандартного вывода задает файл назначения для него. Стандартный вывод записывается и сохраняется в этом файле. После записи никакой информации для пересылки по каналу в другую команду не остается. Переадресация может производиться в конце последовательности программных каналов, но не внутри этой последовательности.

Часть 2. Программирование в BASH-shell

Shell

Командный интерпретатор в среде UNIX выполняет две основные функции:

- представляет интерактивный интерфейс с пользователем, т.е. выдает приглашение, и обрабатывает вводимые пользователем команды;

- обрабатывает и исполняет текстовые файлы, содержащие команды интерпретатора (командные файлы);

В последнем случае, операционная система позволяет рассматривать командные файлы как разновидность исполняемых файлов. Соответственно различают два режима работы интерпретатора: интерактивный и командный.

Существует несколько типов оболочек в мире UNIX. Две главные - это "Bourne shell" и "C shell". Bourne shell (или просто shell) использует командный

синтаксис, похожий на первоначально для UNIX. В большинстве UNIX-систем Bourne shell имеет имя /bin/sh (где sh сокращение от ''shell''). C shell использует иной синтаксис, чем-то напоминающий синтаксис языка программирования Си. В большинстве UNIX-систем он имеет имя /bin/csh.

В Linux есть несколько вариаций этих оболочек. Две наиболее часто используемые, это Новый Bourne shell (Bourne Again Shell) или ''Bash'' (/bin/bash) и Tcsh (/bin/tcsh). Bash - это развитие прежнего shell с добавлением многих полезных возможностей, частично содержащихся в C shell.

Поскольку Bash можно рассматривать как надмножество синтаксиса прежнего shell, любая программа, написанная на sh shell должна работать и в Bash. Tcsh является расширенной версией C shell. При входе в систему пользователю загружается командный интерпретатор по умолчанию. Информация о том, какой интерпретатор использовать для конкретного пользователя находится в файле /etc/passwd.

Возможно, вам захочется выполнить сценарий, написанный для одного из shell Linux, в то время как вы работаете в другом. Предположим, вы работаете в TCSH-shell и хотите выполнить написанный в BASH сценарий, содержащий команды этого (второго) shell. Сначала нужно с помощью команды sh перейти в BASH-shell, выполнить сценарий, а затем вернуться в TCSH. Эту процедуру можно автоматизировать, поставив первыми в сценарии символы #! и указав после них путевое имя программы нужного shell в вашей системе.

Shell всегда изучает первые символы сценария и на их основании делает вывод о том, к какому типу shell этот сценарий относится - BASH, PDKSH или TCSH. Если первый символ - пробел, это сценарий BASH-shell или PDKSH-shell. Если первый символ - знак #, это сценарий TCSH-shell. Если первые символы - #!, то shell читает указанное за ними имя программы. После символов #! всегда должно следовать путевое имя программы нужного shell, по которому можно идентифицировать его тип. Если вы запускаете сценарий из shell, отличного от того, который указан в первой строке запускаемого сценария, то будет вызван shell, указанный в первой строке, и в нем выполнится ваш сценарий. В такой ситуации одного пробела или знака 41 для указания того, что это сценарий BASH или TCSH, бывает недостаточно. Такая идентификация работает только в собственных сценариях этих shell. Чтобы обозначить сценарий другого shell, необходимо поставить символы #! и путевое имя.Например, если поставить в начало первой строки сценария hello комбинацию символов #!/bin/sh, то этот сценарий можно будет выполнять непосредственно из TCSH-shell. Сначала сценарий осуществит переход в BASH, выполнит его команды, а затем вернется в TCSH (или в тот shell, из которого он выполнялся). В следующем примере сценарий hello содержит команду #!/bin/sh. Пользователь выполняет сценарий, находясь в TCSH-shell.

Командные файлы

Командный файл в Unix представляет собой обычный текстовый файл, содержащий набор команд Unix и команд Shell. Для того чтобы командный интерпретатор воспринимал этот текстовый файл, как командный необходимо установить атрибут на исполнение.

```
$ echo "ps -af" > commandfile
$ chmod +x commandfile
```

```
$ ./commandfile
```

В представленном примере команда echo "ps -af" > commandfile создаст файл с одной строкой "ps -af", команда chmod +x commandfile установит атрибут на исполнение для этого файла, команда ./commandfile осуществит запуск этого файла.

Переменные shell

Имя shell-переменной - это начинающаяся с буквы последовательность букв, цифр и подчеркиваний. Значение shell-переменной - строка символов.

Например: Var = "String" или Var = String

Команда echo $Var выведет на экран содержимое переменной Var т.е. строку "String" , на то что мы выводим содержимое переменной указывает символ "$". Так команда echo Var выведет на экран просто строку "Var" .

Еще один вариант присвоения значения переменной Var = 'набор команд Unix'. Обратные кавычки говорят о том, что сначала должна быть выполнена заключенная в них команда), а результат ее выполнения, вместо выдачи на стандартный выход, приписывается в качестве значения переменной.

```
CurrentDate = 'date'
```

Переменной CurrentDate будет присвоен результат выполнения команды date. Можно присвоить значение переменной и с помощью команды "read", которая обеспечивает прием значения переменной с (клавиатуры) дисплея в диалоговом режиме.

```
echo "Введите число"
read X1
echo "вы ввели -" $X1
```

Несмотря на то, что shell-переменные в общем случае воспринимаются как строки, т. е. "35" - это не число, а строка из двух символов "3" и "5", в ряде случаев они могут интерпретироваться иначе, например, как целые числа.

Разнообразные возможности имеет команда "expr".

```
x=7
y=2

rez=expr $x + $y
echo результат=$rez
```

выдаст на экран результат=9

Параметры командного файла

В командный файл могут быть переданы параметры. В shell используются позиционные параметры (т.е. существенна очередность их следования). В командном файле соответствующие параметрам переменные (аналогично shell-переменным) начинаются с символа "$", а далее следует одна из цифр от 0 до 9: При обращении к параметрам перед цифрой ставится символ доллара "$" (как и при обращении к переменным):

$0 соответствует имени данного командного файла;
$1 первый по порядку параметр;
$2 второй параметр и т.д.

Поскольку число переменных, в которые могут передаваться параметры, ограничено одной цифрой, т.е. 9-ю ("0", как уже отмечалось имеет особый смысл), то для передачи большего числа параметров используется специальная команда "shift".

Команда "set" устанавливает значения параметров. Это бывает очень удобно. Например, команда "date" выдает на экран текущую дату, скажем, "Mon May 01 12:15:10 2002", состоящую из пяти слов, тогда set `dateècho $1 $3 $5 выдаст на экран Mon 01 2002

Программные структуры

Как во всяком процедурном языке программирования в языке shell есть операторы. Ряд операторов позволяет управлять последовательностью выполнения команд. В таких операторах часто необходима проверка условия, которая и определяет направление продолжения вычислений.

Команда test

Команда test проверяет выполнение некоторого условия. С использованием этой (встроенной) команды формируются операторы выбора и цикла языка shell. Два возможных формата команды:

test условие
или
[условие]
В shell используются условия различных "типов".
Условия проверки файлов:
-f file
файл "file" является обычным файлом;
-d file
файл "file" - каталог;
-c file
файл "file" - специальный файл;
-r file
Имеется разрешение на чтение файла "file";
-w file
Имеется разрешение на запись в файл "file";

-s file
файл "file" не пустой.
Условия проверки строк:
str1 = str2
строки "str1" и "str2" совпадают;
str1 != str2
строки "str1" и "str2" не совпадают;
-n str1
строка "str1" существует (непустая);
-z str1

строка "str1" не существует (пустая).

Условия сравнения целых чисел:

x -eq y
"x" равно "y",
x -ne y
"x" неравно "y",
x -gt y
"x" больше "y",
x -ge y
"x" больше или равно "y",
x -lt y
"x" меньше "y",
x -le y
"x" меньше или равно "y".

То есть в данном случае команда "test" воспринимает строки символов как целые (!) числа. Поэтому во всех остальных случаях "нулевому" значению соответствует пустая строка. В данном же случае, если надо обнулить переменную, скажем, "x", то это достигается присваиванием "x=0". Сложные условия реализуются с помощью типовых логических операций:

! (not) инвертирует значение кода завершения.
-o (or) соответствует логическому "ИЛИ".
-a (and) соответствует логическому "И"

Управляющие структуры

С помощью управляющих структур пользователь может осуществлять контроль над выполнением Linux-команд в программе. Управляющие структуры позволяют повторять команды и выбирать для выполнения команды, необходимые в конкретной ситуации. Управляющая структура состоит из двух компонентов: операции проверки и команд. Если проверка считается успешной, то выполняются команды. Таким образом, с помощью управляющих структур можно принимать решения о том, какие команды следует выполнять.

Существует два вида управляющих структур: циклы и условия. Цикл используется для повторения команд, тогда как условие обеспечивает выполнение команды при соблюдении некоторых критериев. В BASH-shell используются три вида циклических управляющих структур (while, for и for-in) и две условные управляющие структуры (if и case).

Управляющие структуры while и if - это структуры общего назначения. Они используются, например, для выполнения итераций и принятия решений на основании различных проверок. Управляющие структуры case и for - более специализированные. Структура case представляет собой модифицированную форму условия if и часто используется для построения меню. Структура for - это цикл ограниченного типа. Здесь обрабатывается список значений, и на каждой итерации цикла переменной присваивается новое значение.

В управляющих структурах if и while проверка построена на выполнении Linux-команды. Все команды ОС Linux после выполнения выдают код завершения. Если команда выполнена нормально, выдается код 0. Если по какой-либо причине команда не выполняется, то выдается положительное число, обозначающее тип

отказа. Управляющие структуры if и while проверяют код завершения Linux-команды. Если код - 0, выполняются действия одного типа, если нет - другого.

Дополнительная информация

Во-первых, обязательно обратитесь к **man bash**.

Во-вторых, можно воспользоваться следующим ресурсом.

В-третьих, краткая памятка по командам, которые вам могут понадобиться:

`pwd`

Вывести текущую директорию.

`hostname`

Вывести или изменить сетевое имя машины.

`whoami`

Ввести имя под которым я зарегистрирован.

`date`

Вывести или изменить дату и время. Например, чтобы установить дату и время равную 2000-12-31 23:57, следует выполнить команду:

date 123123572000

`time`

Получить информацию о времени, нужного для выполнения процесса + еще кое-какую информацию. Не путайте эту команду с date. Например: Я могу определить как много времени требуется для вывода списка файлов в директории, набрав последовательность:

time ls

`who`

Определить кто из пользователей работает на машине.

`finger [имя_пользователя]`

Системная информация о зарегистрированном пользователе. Попробуйте: finger root.

`uptime`

Количество времени прошедшего с последней перезагрузки.

`ps –a`

Список текущих процессов.

`top`

Интерактивный список текущих процессов отсортированных по использованию cpu.

`uname`

Вывести системную информацию.

`free`

Вывести информацию по памяти.

`df –h`

(=место на диске) Вывести информацию о свободном и используемом месте на дисках (в читабельном виде).

`du / -bh | more`

(=кто сколько занял) Вывод детальной информации о размере файлов по директориям начиная с корневой (в читабельном виде).

`cat /proc/cpuinfo`

Системная информация о процессоре. Заметьте, что файла в /proc директории - не настоящие файлы. Они используются для получения информации, известной системе.

`cat /proc/interrupts`

Используемые прерывания.

`cat /proc/version`

Версия ядра Linux и другая информация

`cat /proc/filesystems`

Вывести используемые в данный момент типы файловых систем.

`cat /etc/printcap`

Вывести настройки принтера.

`set | more`

Вывести текущие значения переменных окружения.

`echo $PATH`

Вывести значение переменной окружения "PATH" Эта команда может использоваться для вывода значений других переменных окружения. Воспользуйтесь командой set, для получения полного списка.

`grep …`

Поиск вхождения регулярного выражения в строки заданного файла (потока).

GREP

Одним из известных инструментов поиска в Unix-подобных системах, которые можно использовать для поиска всего, будь то файл или строка или несколько строк в файле, – это утилита grep. Он очень обширен в функциональности, за счет большого количества поддерживаемых им опций, таких как: поиск с использованием строкового шаблона или регулярных выражений шаблон или perl based регулярных выражений и т.д.

Из-за его различных функциональных возможностей инструмент grep имеет множество вариантов, включая egrep (Extended GREP), fgrep (Fixed GREP), pgrep (Process GREP), rgrep (рекурсивный GREP) и т.д. Но эти варианты имеют незначительные отличия от оригинального grep.

Разберемся с различиями между тремя основными вариантами: «grep», «egrep» и «fgrep». Чтобы пользователи Linux знали, что выбирать в соответствии со своими требованиями.

Изучим некоторые специальные символы, известные как метасимволы. Они помогают создавать более сложные поисковые выражения:

. будет соответствовать любому символу;
[ ] будет соответствовать диапазону символов;
[^] будет соответствовать всем символам, кроме указанных в фигурных скобках;
* будет соответствовать любому количеству символов, предшествующих звездочке, в том числе нулю;
+ будет соответствовать одному или нескольким из стоящих перед ним выражений;
? будет соответствовать нулю или одному из стоящих перед ним выражений;
{n} будет соответствовать 'n' повторениям предшествующих выражений;
{n,} будет соответствовать не менее 'n' повторениям предшествующих выражений;
{n m} будет соответствовать не менее 'n' и не более 'm' повторениям предшествующих выражений;
{,m} будет соответствовать не более или равному 'm' повторениям предшествующих выражений;
\ является escape-символом (символом экранирования), используемым, когда нужно включить один из метасимволов.

Различия между grep, egrep и fgrep

Для примера создадим файл example со следующим содержанием:

```
This is
a file
Text
(f|g)ile
```

Команда Grep

Grep или Global Regular Expression Print – основная программа поиска в Unix-подобных системах, которая может искать любой тип строки в любом файле или списке файлов или даже выводить любую команду.

В качестве шаблона поиска он использует обычные регулярные выражения, кроме обычных строк. В Basic Regular Expressions (BRE) метасимволы вроде: ‘{‘,

‘}’, ‘(‘, ‘)’, ‘|’, ‘+’, ‘?’ теряют свой смысл и считаются нормальными символами строки и должны быть экранированы с помощью символа ‘\‘, если их следует рассматривать как специальные символы.

Кроме того, grep использует алгоритм Boyer-Moore для быстрого поиска любой строки или регулярного выражения.

```
user@user:~$ grep '(f|g)ile' example
(f|g)ile

user@user:~$ grep '\(f\|g\)ile' example
a file
```

В примере выше, когда команда запускается без экранирования **‘(‘ ‘)’** и **‘|’**, затем **grep** ищет полную строку, то есть «**(f | g) ile**» в файле. Но когда специальные символы были экранированы, **grep** обрабатывает их как метасимволы и ищет слова **«file»** или **«gile»** в файле.

Команда egrep

Egrep или **grep -E** – это другая версия **grep** или **Extended grep.** Эта версия **grep** эффективна и быстра, когда дело доходит до поиска шаблона регулярных выражений, поскольку она обрабатывает метасимволы как есть и не заменяет их как строки. **Egrep** использует **ERE** или **Extended Extended Expression.**

В случае **egrep,** даже если вы не избегаете метасимволы, команда будет относиться к ним как к специальным символам и заменять их своим особым значением вместо того, чтобы рассматривать их как часть строки

```
user@user:~$ egrep '\(f\|g\)ile' example
(f|g)ile

user@user:~$ egrep '(f|g)ile' example
a file
```

Здесь **egrep** ищет строку **«file»,** когда мета-символы не экранированы, поскольку это означает значение этих символов. Но, когда эти символы стали экранированы, **egrep** рассматривает их как часть строки и ищет полную строку «**(f | g) ile**» в файле.

Команда fgrep

Fgrep или **Fixed grep** или **grep -F** – это еще одна версия **grep,** которая необходима, когда дело доходит до поиска всей строки вместо регулярного выражения, поскольку оно не распознает ни регулярные выражения, ни метасимволы. Для поиска любой строки напрямую выбирайте эту версия **grep.**
Fgrep ищет полную строку и не распознает специальные символы как часть регулярного выражения, несмотря на то экранированы символы или нет.

```
user@user:~$ fgrep '(f|g)ile' example
(f|g)ile
```

```
user@user:~$ fgrep '\(f\|g\)ile' example
```

Когда метасимволы не были экранированы, fgrep искала полную строку «(f | g) ile» в файле, а когда метасимволы стали экранированы, команда fgrep искала «\ (f \ | g \) Ile» все символы которые есть в файле.

Задание для выполнения

1. Вывести любое сообщение с помощью команды echo перенаправив вывод:
   - в несуществующий файл с помощью символа >;
   - в несуществующий файл с помощью символа >>;
   - в существующий файл с помощью символа >;
   - в существующий файл с помощью символа >>;

Объяснить результаты.

2. Переадресовать стандартный ввод для команды cat на файл.

3. Вывести сообщение с помощью команды echo в канал ошибок. Создать файл myscript:

```
#!/bin/sh
echo stdout
echo stderr>&2
exit 0
```

Запустить его:

- без перенаправления (sh myscript);
- перенаправив стандартный вывод в файл, просмотреть содержимое файла (sh myscript > file1);
- перенаправить стандартный канал ошибок в существующий и несуществующий файлы с помощью символов > и >> ;
- перенаправив стандартный вывод в файл 1, стандартный канал ошибок - в файл 2;
- перенаправив стандартный вывод и стандартный канал ошибок в файл 3;
- перенаправив стандартный вывод в файл 4 с помощью символа >, а стандартный канал ошибок в файл 4 с помощью символа >>;

Объяснить результаты.

4. Вывести третью и шестую строку из последних пятнадцати строк отсортированного в обратном порядке файла /etc/group.

5. Подсчитать при помощи конвейера команд количество блочных и количество символьных устройств ввода-вывода, доступных в системе.

6. Написать скрипт, выводящий на консоль все аргументы командной строки, переданные данному скрипту. Привести **различные варианты запуска** данного скрипта, в том числе без непосредственного вызова интерпретатора в командной строке.

7. Написать скрипт согласно индивидуальному заданию. Номер варианта согласовать с преподавателем.

Варианты индивидуальных заданий

1. Реализовать командный файл, реализующий меню из трех пунктов (в цикле):

   1) ввести пользователя и вывести на экран все процессы, запущенные данным пользователем;

   2) показать всех пользователей, в настоящий момент, находящихся в системе;

   3) завершение.

2. Реализовать командный файл, реализующий меню из трех пунктов (в цикле):

1) вывести всех пользователей, в настоящее время, работающих в системе;

2) копирование одного файла в другой каталог. <Имя файла> и <Имя каталога> вводятся пользователем с клавиатуры;

3) завершение.

3. Реализовать командный файл, реализующий меню из трех пунктов (в цикле):

1) показать все процессы пользователя, запустившего данный командный файл;

2) послать сигнал завершения процессу текущего пользователя (ввести PID процесса);

3) завершение.

4. Реализовать командный файл, реализующий меню из трех пунктов (в цикле):

1) определить количество запущенных данным пользователем процессов bash (предусмотреть ввод имени пользователя);

2) выводит информацию об имени компьютера, IP- адреса и список всех пользователей зарегистрированных в данный момент на компьютере;

3) завершение.

5. Реализовать меню из трех пунктов (в цикле):

1) поиск файла в каталоге. <Имя файла> и <Имя каталога> вводятся пользователем с клавиатуры;

2) завершить все процессы bash активного пользователя.

3) завершение.

6. Написать командный файл который в цикле по нажатию клавиши выводит информацию о системе, активных пользователях в системе, а для введенного имени пользователя выводит список активных процессов данного пользователя.

7. Реализовать командный файл который при старте выводит информацию о системе, информацию о пользователе, запустившем данный командный файл, далее в цикле выводит список активных пользователей в системе – запрашивает имя пользователя и выводят список всех процессов bash запущенных данным пользователем.

8. Реализовать командный файл, реализующий символьное меню (в цикле):

1) вывод полной информации о файлах каталога: Ввести имя каталога для отображения;

2) изменить атрибуты файла: файл и атрибуты вводятся с клавиатуры по запросу. После изменения атрибутов вывести на экран расширенный список файлов для проверки установленных атрибутов;

3) выход.

9.Реализовать командный файл, реализующий символьное меню (в цикле):

1) вывод полной информации о файлах каталога (ввести имя каталога для отображения);

2) создать командный файл: файл вводится с клавиатуры по запросу, далее изменяются атрибут файла на исполнение, затем вводится с клавиатуры строка, которую будет исполнять командный файл. После изменения

атрибутов вывести на экран расширенный список файлов для проверки установленных атрибутов и запустить созданный командный файл;

3) завершение.

10. Реализовать командный файл, позволяющий в цикле посылать всем активным пользователям сообщение – сообщение вводится с клавиатуры. Командный файл при старте выводит имя компьютера, имя запустившего командный файл пользователя, тип операционной системы, IP-адрес машины.

11. Реализовать командный файл, позволяющий в цикле посылать всем активным пользователям (исключая пользователя, запустившего данный командный файл) сообщение – сообщение вводится с клавиатуры. Командный файл при старте выводит имя компьютера, имя запустившего командный файл пользователя, тип операционной системы, список загруженных модулей.

12. Реализовать командный файл который при старте выводит информацию о системе, информацию о пользователе, запустившем данный командный файл, далее в цикле выводит список активных пользователей в системе – запрашивает имя пользователя и выводят список всех терминалов, на которых зарегистрирован этот пользователь.

13. Реализовать командный файл, который выводит: дату, информацию о системе, текущий каталог, текущего пользователя, настройки домашнего каталога текущего пользователя, далее в цикле выводит список активных пользователей – запрашивает имя пользователя и выводит информацию об активности данного пользователя.

14. Реализовать командный файл, который выводит: дату в формате день:месяц:год – время, информацию о системе в формате: имя компьютера : версия ОС : IP адрес : имя текущего пользователя : текущий каталог, выводит домашний каталог текущего пользователя и основные переменные окружения. Далее в цикле выводит список активных пользователей – запрашивает имя пользователя и выводит информацию об активности введенного пользователя.

15. Написать командный файл с использованием цикла for, выводящий на консоль размеры и права доступа для всех файлов в заданном каталоге и всех его подкаталогах (имя каталога задается пользователем в качестве первого аргумента командной строки).

16. Написать командный файл, находящий в заданном каталоге и всех его подкаталогах все файлы, владельцем которых является заданный пользователь. Имя владельца и каталог задаются пользователем в качестве первого и второго аргумента командной строки.

17. Написать командный файл поиска одинаковых по их содержимому файлов в двух каталогах, например, Dir1 и Dir2. Пользователь задает имена Dir1 и Dir2 в качестве первого и второго аргумента командной строки. На экран выводятся число просмотренных файлов и результаты сравнения.

18. Реализовать командный файл, реализующий символьное меню (в цикле):

1) копирование файлов: в этом пункте выводится информация о содержимом текущего каталога, далее предлагается интерфейс копирования файла: ввод имени файла и ввод каталога для копирования. По выполнению пункта выводится содержимое каталога, куда был скопирован файл и выводится содержимое скопированного файла;

2) завершение.

19. Реализовать командный файл, реализующий символьное меню (в цикле):

1) поиск всех файлов в заданном каталоге, у которых нет прав на редактирование;
2) поиск файлов в заданном каталоге по заданным правам;
3) завершение.

20. Реализовать командный файл, реализующий символьное меню (в цикле):
1) поиск файлов в заданном каталоге, размер которых не превосходит заданного параметра;
2) поиск файлов в заданном каталоге по заданному времени создания.
3) завершение.

Отчет

Как и в других работах, отчет по проделанной работе представляется преподавателю в стандартной форме: на листах формата А4, с титульным листом (включающим тему, фио, номер зачетки и пр.), целью, ходом работы и выводами по выполненной работе. Каждое задание должно быть отражено в отчете следующим образом: 1) что надо было сделать, 2) как это сделали, 3) что получилось, и, в зависимости от задания – 4) почему получилось именно так, а не иначе. При наличии заданий с вариантами необходимо указать свой вариант, и расчет его номера, а также все вышеуказанное для данного варианта задания.