Лабораторная работа №4

Тема: СОЗДАНИЕ И ВЫПОЛНЕНИЕ КОМАНДНЫХ ФАЙЛОВ
В СРЕДЕ ОС LINUX

Цель работы: изучить методы создания и выполнения командных файлов на языке Shell - интерпретатора.

Теоретические сведения

В предыдущих лабораторных работах взаимодействие с командным интерпретатором Shell осуществлялось с помощью командной строки. Однако, Shell является также и языком программирования, который применяется для написания командных файлов (shell - файлов). Командные файлы также называются скриптами и сценариями. Shell - файл содержит одну или несколько выполняемых команд (процедур), а имя файла в этом случае используется как имя команды.

bash-скрипт должен начинаться со строки:
#!/bin/bash

Для создания сценариев можно использовать любой текстовый редактор. Создайте новый файл и введите следующие строки.

```
#!/bin/bash
echo "It is my first shell-file!!!"
# комментарий
ls $HOME
echo "Done!"
```

Первая строка указывает текущей оболочке, какую программу следует использовать для интерпретации файла. В данном случае это оболочка bash. Это сделано для того, что если сценарий вызывается в пределах другой оболочки или файлового менеджера - они будут "знать", что для выполнения этого сценария требуется оболочка bash.

Вторая строка - это первая команда в сценарии. Команда *echo* используется для вывода на экран простой информационной строки.

Третья строка - комментарий.

Далее следуют ещё две команды - команда *ls*, которая берёт имя каталога в качестве параметра, и, наконец, команда *echo*, выводящая на экран информацию об успешном выполнении работы.

Для запуска скрипта напишите в терминале:
sh имя_скрипта или **source ./**имя_скрипта

Другой способ запуска с предварительным назначением прав:
Для того чтобы скрипт стал исполняемым, могут быть использованы следующие команды:

chmod +rx scriptname # *выдача прав на чтение/исполнение любому пользователю*
chmod u+rx scriptname # *выдача прав на чтение/исполнение только "владельцу" скрипта*
Запуск: **./scriptname**

Переменные командного интерпретатора

Как любой язык программирования, командный язык shell поддерживает переменные. Тип их — строковый. Для обозначения переменных Shell используется последовательность букв, цифр и символов подчеркивания; переменные не могут начинаться с цифры.

Оператор присваивания выглядит так:

```
$имя_переменной=значение
```

Имя должно начинаться с буквы и может состоять из латинских букв, цифр, знака подчеркивания. Если значение переменной содержит специальные символы, в имени файла, то при указании его имени в команде этот символ нужно экранировать знаком «\» (обратный слэш) или заключать все имя в двойные кавычки.

Вот ряд символов, которые имеют специальное значение для командного интерпретатора, и их использование не рекомендуется:

~ ! @ # $ & % * ( ) [ ] { } ' " \ : ; > < пробел

Внимание!!! Обратите внимание, что переменные в shell чувствительны к регистру. Например, Myvar и myvar – это имена разных переменных.

Позиционные переменные.

Переменные вида $n, где n - целое число, используются для идентификации позиций элементов в командной строке с помощью номеров, начиная с нуля. Например, в командной строке

```
cat text_1 text_2...text_9
```

аргументы идентифицируются параметрами $1...$9. Для имени команды всегда используется S0. В данном случае $0 - это cat, $1 - text_1, $2 - text_2 и т.д. Для присваивания значений позиционным переменным используется команда **set**, например:

```
set arg_1 arg_2... arg_9
```

здесь $1 присваивается значение аргумента arg_1, $2 - arg_2 и т.д.

Вывод переменных.

Операция подстановки значения переменной обозначается символом $. Вывести значение переменной можно командой *echo*:

```
$ var="Это моя переменная!"
$ echo var      # выводит имя переменной var
$ echo $var     # выводит значение переменной "Это моя переменная!"
```

Ставя перед именем переменной знак **$**, мы сообщаем интерпретатору, что нужно заменить ее значением. Это называется подстановкой переменной. Но что будет, если мы попробуем сделать так:

```
$echo foo$var  #программа выведет "foo"
```

Мы хотели вывести на экран надпись '*fooЭто моя переменная!*', но ничего не получилось. Что же произошло? Интерпретатор не смог определить значение какой именно переменной нужно подставить ($v, $va, $var и т.д.) В таких неоднозначных случаях можно явно указать на имя переменной:

```
$echo foo${var}
```

Для получения информации обо всех аргументах (включая последний) используют метасимвол *. Пример:

```
echo $*
```

Внимание!!! Для вывода переменных можно использовать команду *printf*:

```
PI=3,14159265358979
printf "Число пи с точностью до 2 знака после запятой = %1.2f" $PI
printf "Число пи с точностью до 9 знака после запятой = %1.9f" $PI
```

Арифметические операции

Команда *expr* (express -- выражать) вычисляет выражение expression и записывает результат в стандартный вывод. Элементы выражения разделяются пробелами; символы, имеющие специальный смысл в командном языке, нужно экранировать. Строки, содержащие специальные символы, заключают в апострофы. Используя команду *expr*, можно выполнять сложение, вычитание, умножение, деление, взятие остатка, сопоставление символов и т. д.

Пример. Сложение, вычитание:

```
b=190
a=` expr 200 - $b`
```

где ` - обратная кавычка (левая верхняя клавиша). Умножение *, деление /, взятие остатка %:

```
d=` expr $a + 125 "*" 10`
c=` expr $d % 13`
```

Здесь знак умножения заключается в двойные кавычки, чтобы интерпретатор не воспринимал его как метасимвол. Во второй строке переменной *c* присваивается значение остатка от деления переменной *d* на 13.

Сопоставление символов с указанием числа совпадающих символов:

```
concur=` expr "abcdefgh" : "abcde"`
echo $concur
ответ 5.
```

Операция сопоставления обозначается двоеточием (:). Результат - переменная concur.

Подсчет числа символов в цепочках символов. Операция выполняется с

использованием функции *length* в команде *expr*:

```
chain="The program is written in Assembler"
str=` expr length "$chain"`
Echo $str
```

ответ 35. Здесь результат подсчета обозначен переменной *str*.
Внимание!!! Возможно использование следующей формы записи:

```
echo "Мне $[2000-1957] лет."
```

Логические выражения и операторы управления

Условия, связанные с файлами.

-b файл	Файл существует и является блочным специальным
-c файл	Файл существует и является символьным специальным
-d файл	Файл существует и является каталогом
-e файл	Файл существует
-f файл	Файл существует и является обычным файлом
-h файл	Файл существует и является символической связью
-r файл	Файл существует и доступен текущему пользователю для чтения
-s файл	Файл существует и имеет ненулевой размер
-t хэндл	Хэндл открыт и соответствует терминалу
-w файл	Файл существует и доступен текущему пользователю для записи
-x файл	Файл существует и доступен текущему пользователю для выполнения
-O файл	Файл существует и принадлежит текущему пользователю
-G файл	Файл существует и принадлежит группе текущего пользователя
-N файл	Файл существует и был изменен после последнего чтения
файл1 –nt файл2	Файл 1 новее (т.е. был изменен позднее), чем файл 2
файл1 –ot файл2	Файл 1 старее (т.е. был изменен раньше), чем файл 2
файл1 –ef файл2	Оба имени ссылаются на один и тот же файл (как жесткие ссылки)

Условия, связанные со строками.

-z строка	Строка пуста (имеет нулевую длину)
-n строка	Строка непустая
строка	Строка непустая
строка1 = строка2	Строки равны
строка1 == строка2	Строки равны
строка1 != строка2	Строки не равны
строка1 < строка2	Строка 1 меньше (в словарном смысле), чем строка 2
строка1 > строка2	Строка 1 больше (в словарном смысле), чем строка 2

Условия, связанные с числами.

число1 ОП число2	Операция сравнения ОП может быть любой из следующих: *-eq (равно),* *-ne (не равно),* *-lt (меньше),* *-le (меньше либо равно),* *-gt (больше),* *-ge (больше либо равно).*

Логические связки

! выражение	Инверсия (логическое отрицание)
выраж1 -a выраж2	Конъюнкция (логическое И)
выраж1 -o выраж2	Дизъюнкция (логическое ИЛИ)

Встроенные функции.

Встроенные команды являются частью интерпретатора и не требуют для своего выполнения проведения последовательного поиска файла команды и создания новых процессов.

Функция	**Описание**
cd	Команда изменяет текущий каталог.
pwd	Выводит название текущего рабочего каталога
eval	Конструирование команды на лету, из указанных аргументов, и отправка ее на выполнение.
SECONDS	Содержит время работы скрипта в секундах
USER	Содержит имя пользователя
UID	Числовой идентификатор текущего пользователя
read	«Читает» значение введенной переменной. … *read var1 # Обратите внимание -- перед именем переменной отсутствует символ '$'* *echo "var1 = $var1"* … Команда *read* имеет ряд очень любопытных опций, которые позволяют выводить подсказку - приглашение ко вводу (prompt), и даже читать данные не дожидаясь нажатия на клавишу **ENTER**. *-s -- подавляет эхо-вывод, т.е. ввод с клавиатуры не отображается на экране.* *-n N -- ввод завершается автоматически, сразу же после ввода N-го символа.* *-p -- задает вид строки подсказки - приглашения к вводу (prompt).*

set	Команда изменяет значения внутренних переменных сценария.
unset	Команда удаляет переменную, фактически -- устанавливает ее значение в *null*.
export	Команда экспортирует переменную, делая ее доступной дочерним процессам.
declare, typeset	Команды *declare* и *typeset* задают и/или накладывают ограничения на переменные. declare -r -- делает переменную доступной только для чтения (аналог const в C)
Exit	Безусловное завершение работы сценария. Команде можно передать целое число, которое будет возвращено вызывающему процессу как код завершения. Вообще, считается хорошей практикой завершать работу сценария, за исключением простейших случаев, командой *exit 0*, чтобы проинформировать родительский процесс об успешном завершении.
exec	Это встроенная команда интерпретатора shell, заменяет текущий процесс новым процессом, запускаемым командой *exec*. Обычно, когда командный интерпретатор встречает эту команду, то он порождает дочерний процесс, чтобы исполнить команду. При использовании встроенной команды *exec*, оболочка не порождает еще один процесс, а заменяет текущий процесс другим. Для сценария это означает его завершение сразу после исполнения команды *exec*. По этой причине, если вам встретится *exec* в сценарии, то, скорее всего это будет последняя команда в сценарии.
wait	Останавливает работу сценария до тех пор пока не будут завершены все фоновые задания или пока не будет завершено задание/процесс с указанным номером задания. Возвращает код завершения указанного задания/процесса. Вы можете использовать команду *wait* для предотвращения преждевременного завершения сценария до того, как завершит работу фоновое задание.
kill	Принудительное завершение процесса путем передачи ему соответствующего сигнала. **kill $$ # Сценарий завершает себя сам.**
sh <filename.sh>	Вызов файла filename.sh
trap [cmd] [cond]	Перехват сигналов прерывания, где: cmd - выполняемая команда; cond=0 или EXIT - в этом случае команда cmd выполняется при завершении интерпретатора; cond=ERR - команда cmd выполняется при обнаружении ошибки; cond - символьное или числовое обозначение сигнала, в этом случае команда cmd выполняется при приходе этого сигнала;
umask [-o | -s] [nnn]	Устанавливает маску создания файла (маску режимов доступа создаваемого файла, равную восьмеричному числу nnn: 3 восьмеричных цифры для пользователя, группы и других). Если аргумент nnn отсутствует, то команда сообщает текущее значение маски. При наличии флага -o маска выводится в восьмеричном виде, при наличии флага -s - в символьном представлении;
ls[ключи] параметры	Для каждого параметра-каталога выдает информацию о файлах

	этого каталога. Для параметра-файла выдает информацию о данном файле. Если параметров нет, выдает информацию о текущем каталоге. Ключи: *-l - Подробный формат, по одному файлу в строке. Указываются: тип файла, права доступа для владельца, группы-владельца и прочих пользователей, количество жестких связей, владелец, группа-владелец, размер в байтах, дата и время последнего изменения, имя.* *-a - Включаются данные об элементах каталога . и ..* *-i - Включается номер индексного дескриптора (inode).* *-d - Для каталога выводятся данные о самом каталоге, а не о его содержимом.* *-t - Сортировка по времени последнего изменения файла.* *-u - Сортировка по времени последнего обращения к файлу.* *-c - Сортировка по времени создания файла.* *-r - Обратный порядок сортировки*
who	Выводит список пользователей, работающих в данный момент в системе (в пределах домена локальной сети). Для каждого пользователя выводится одна строка, включающая его имя входа, имя терминала, время входа в систему и имя компьютера.
history [-c] [число]	Выдает «список истории», содержащий последние введенные команды (по умолчанию – до 500 строк). С ключом -c очищает список истории. С аргументом – положительным числом N выдает только последние N строк истории.
hostname [ключи]	Выдает на стандартный вывод имя данного компьютера. Чтобы получить полное интернетовское имя (Fully Qualified Domain Name), надо указать ключ –-fqdn.
times	Выдает суммарное процессорное время, затраченное в режиме ядра и в режиме задачи данным шеллом и всеми процессами, запускавшимися из шелла.
ps[ключи]	Выдает информацию о процессах, выполняющихся в системе. По умолчанию выдает информацию только о процессах, запущенных с данного терминала, причем для каждого процесса выдаются 4 поля данных: идентификатор процесса (PID), имя терминала (TTY), затраченное процессорное время (TIME), имя выполняемой команды (CMD). Если процесс не прикреплен ни к какому терминалу, вместо имени терминала выдается знак ?. При вызове с ключом -f дополнительно выдаются идентификатор пользователя (UID), идентификатор процесса-родителя (PPID), приоритет (C), время запуска (STIME), а для выполняемой команды указываются полное имя и параметры. При использовании ключа -e перечисляются все процессы, в том числе запущенные системой или другими пользователями. Имеется еще очень много ключей, позволяющих изменить объем и формат выдаваемой информации о процессах.
uname [ключи]	Выводит строку информации о системе, согласно следующим ключам: *-s – Имя ОС* *-n – Сетевое имя компьютера* *-v – Версия ОС* *-r – Номер выпуска ОС* *-m – Тип компьютера*

	-a – Вся перечисленная выше информация Отсутствие ключей эквивалентно заданию ключа **–s.**
date [ключи] [новая_дата]	Выдает системную дату и время либо устанавливает новые дату и время. Ключи позволяют изменить формат выдачи. Новая дата и время задаются в формате: ММДДЧЧмм[[СС]ГГ][.сс] (месяц, день, часы, минуты, столетие, год, секунды).
cal [ключи] [месяц [год]]	Выдает табель-календарь на указанный месяц. С ключом –y выдает календарь на весь год. При заданном ключе –m первым днем недели считается понедельник (по умолчанию неделя начинается с воскресенья).
du [ключи] [список_имен]	Выдает суммарные данные об использовании дискового пространства файлами, содержащимися в указанных каталогах и их подкаталогах. Некоторые ключи приведены в следующей таблице: *-a – Выдает данные не только о каталогах, но и о каждом файле* *-b – Выдает объем дисковой памяти в байтах (по умолчанию выдается число блоков)* *-c – Формирует общую сумму* *-S – Не включает в размер каталога размеры его подкаталогов* *-s – Для каждого параметра-каталога выдается только общая сумма, без подкаталогов.*
stat [список_имен]	Выдает информацию о файле, содержащуюся в дескрипторе файла (inode). Формат выдачи ориентирован скорее на восприятие человеком, чем на дальнейшую обработку (информация выдается в несколько строк, с названиями полей).
finger [ключи] [список_имен]	Выдает доступную информацию о перечисленных в списке пользователях системы. Если список не задан, выдает информацию обо всех пользователях, работающих в системе в данный момент. Если задан ключ –s, выдает минимальный набор информации в одной строке. Ключ –l задает подробный многострочный формат информации. Если ключи не заданы, то при заданном списке имен используется формат –l, а если список не задан, то формат –s.
cd [путь] или chdir [путь]	Устанавливает каталог по указанному пути в качестве текущего. Если путь не указан, использует «домашний» каталог пользователя, полное имя которого хранится в переменной HOME.
cp файл1 файл2 или cp файлы каталог	В первой форме – копирует параметр файл1 в файл2. Во второй форме – копирует один или несколько файлов в указанный каталог.
ln [-s] новое_имя файл	Без ключа – создает жесткую связь с файлом, т.е. дает существующему файлу дополнительное имя, в том же или в другом каталоге. Имя может содержать путь к каталогу. Счетчик связей файла увеличивается на 1. С ключом -s создает символическую связь, т.е. новый файл, содержащий полное имя существующего файла (аналог ярлыка Windows). Счетчик связей файла при этом не увеличивается.
mv файл1 файл2 или mv список_файлов каталог	В первой форме – переименовывает файл1 в файл2 (или перемещает в другой каталог). Во второй форме – перемещает один или несколько файлов в указанный каталог.

rename старое_имя новое_имя	Переименовывает файл или каталог.
rm [ключи] файлы	Удаляет указанные файлы, точнее – удаляет из каталогов имена и уменьшает на 1 счетчики связей файлов. Действительному удалению подлежат только файлы, для которых число связей стало равным 0 (т.е. удалены все имена файла). Ключ -r позволяет удалять целые каталоги вместе с их содержимым. Ключ -i требует от системы задавать для каждого файла вопрос о необходимости его удаления.
mkdir список_каталогов	Создает один или несколько пустых каталогов с заданными именами. Каждый пустой каталог содержит, тем не менее, два имени: имя . описывает сам каталог, а имя .. описывает родительский каталог.
chmod права список_файлов	Позволяет изменить права доступа к заданным файлам (или каталогам). Изменяемые права могут быть заданы двумя способами: либо в символьном виде, либо с помощью трех восьмеричных цифр. Символьное задание прав состоит из трех элементов: категория пользователей, для которой задаются права (u – владелец файла, g – группа-владелец, o – прочие пользователи, a – все пользователи), выполняемая операция (+ – добавить право, - – отменить право, = – присвоить только это право, отменив остальные права) и конкретное право (r – чтение, w – запись, x – выполнение). Можно указать несколько категорий пользователей и несколько операций с разными правами для одной категории. Можно также в одной команде задать разные права для разных категорий пользователей, разделив их запятыми. Например, запись ug+r-w,o=x означает: «Для владельца и группы разрешить чтение и запретить запись (право на выполнение не менять), для остальных пользователей разрешить выполнение, запретить чтение и запись». Второй способ задания прав предполагает явное задание всех прав в виде восьмеричного числа из трех цифр. Первая цифра задает три бита прав для владельца, вторая цифра – для группы, третья – для прочих. Например, число 751 означает набор прав, который команда ls -l отобразила бы в виде rwxr-x--x, т.е. все права для владельца, чтение и выполнение для группы, только выполнение для прочих.
find список_каталогов [ключи]	Служит для поиска файлов с известным именем и/или другими атрибутами в дереве файловой системы. Заданный каталог или несколько каталогов определяют части файловой системы, в которых ведется поиск. Выполняется просмотр подкаталогов всех уровней, начиная с заданного каталога. Ключи определяют условия поиска файлов и действия с найденными файлами. Некоторые ключи приведены в таблице. Если задано более одного условия, проверяется истинность всех (конъюнкция). В отличие от большинства других команд, ключи задаются не одной буквой, а целым словом.
name имя_файла	Истина, если имя файла (без пути) совпадает с заданным.
perm 8-ричное_число	Истина, если права доступа совпадают с заданными (см. chmod)
atime число_дней	Истина, если к файлу были обращения за последние дни.

mtime число_дней	Истина, если файл был изменен за последние дни.
newer файл	Истина, если файл «новее», чем указанный файл, т.е. был изменен позднее.
type символ	Истина для всех файлов указанного типа (f – обычный файл, d – каталог, b – блочное устройство, c – символьное устройство, p – именованный канал, s – символическая связь).
group имя_группы	Истина, если имя группы-владельца совпадает с заданным.
print	Не проверяет никаких условий, а лишь указывает, что нужно выдать на стандартный вывод полные имена найденных файлов.
mkdir [ключи] каталог	Создает каталог с указанным именем. Если ключи не заданы, то требуется, чтобы уже существовал родительский каталог. При заданном ключе –p команда может создать сразу несколько вложенных каталогов.
rmdir [ключи] список_каталогов	Удаляет перечисленные каталоги. Удаляемый каталог должен быть пустым. При заданном ключе –p команда будет также удалять родительские каталоги, если они становятся пусты. Для удаления непустого каталога вместе с его файлами можно использовать команду rm -r.
cat список_файлов	Читает файлы-параметры и копирует их содержимое на стандартный вывод. Если параметры не заданы, просто передает стандартный ввод на стандартный вывод.

Команды для работы с данными

echo [ключи] параметры

Копирует свои параметры на стандартный вывод (но с учетом специальных символов, если они имеются). Если не задан ключ –n, то в конце выдачи добавляется перевод строки.

Если задан ключ –e, то в выдаваемой строке можно использовать обозначения некоторых «непечатных» символов с помощью знака \. В частности, \n означает перевод строки, \t – символ табуляции, \a – звонок, а \nnn, где nnn – от одной до трех восьмеричных цифр, или \xnnn (nnn – от одной до трех шестнадцатеричных цифр) означает соответствующий символ кода ASCII.

more [файл]

Простая, но полезная команда, которая выводит файл-параметр (или, в его отсутствие, стандартный ввод) порциями, умещающимися на экране. Для вывода следующей порции нужно нажать клавишу «пробел».

less [файл]

Более современная команда просмотра файла (имя команды явно пародирует more). Позволяет, в частности, перемещаться по файлу вперед и назад. Для выдачи сводки по командам перемещения следует ввести h, для выхода из просмотра нужно ввести q.

wc [ключи] [файлы]

Для каждого параметра-файла (или для стандартного ввода) выдается строка, содержащая, в зависимости от ключа, число строк в файле (ключ -l), число слов (ключ -w) или число символов (ключ -c). По умолчанию (без ключей) выдаются все три числа.

head [ключи] [файл]

Выдает указанное число первых строк файла-параметра или стандартного ввода. По умолчанию выдаются 10 строк. Ключ –n число указывает иное число строк. Ключ –c размер указывает, что вместо определенного числа строк следует выдать указанное число начальных байтов, при этом размер можно также указывать в килобайтах (для этого запись размера нужно завершить суффиксом k), в мегабайтах (суффикс m) или в стандартных блоках по 512 байт (суффикс b).

tail [ключи] [файл]

Выдает на стандартный вывод несколько последних строк файла-параметра или стандартного ввода. По умолчанию выдаются 10 строк. Ключи –n число и –c размер действуют так же, как для команды head. Если перед числом строк или байтов записан знак +, то соответствующее число указывает, сколько надо пропустить от начала файла, в противном случае – сколько нужно оставить в конце файла.

grep [ключи] образец [список_файлов]

Выполняет поиск заданной строки-образца в указанных файлах или в стандартном вводе. Если образец содержит пробелы, его следует заключить в кавычки. По умолчанию на стандартный вывод выдаются все строки, содержащие образец. Если проверяется несколько файлов, то перед выводимыми строками выдается имя файла.

При задании образца можно использовать регулярные выражения, задающие шаблон поиска строки. Синтаксис и семантика регулярных выражений напоминают использование подстановочных знаков при поиске в Microsoft Word.

Основные символы, используемые при записи регулярных выражений, приведены в таблице.

.	Любой символ.
\w	Любая буква или цифра.
\W	Любой символ, кроме букв и цифр
[символы]	Любой из перечисленных символов. Можно задавать диапазоны через знак - (например, [0-9] соответствует любой цифре).
[^символы]	Любой символ, кроме перечисленных. Например, [^A-Za-z] означает любой символ, кроме латинских букв.
^	Начало строки.
$	Конец строки.
выраж*	Выражение присутствует ноль или более раз.
выраж?	Выражение присутствует ноль или один раз.
выраж+	Выражение присутствует один или более раз.
выраж1выраж2	Последовательное соединение строк, соответствующих выражениям выраж1 и выраж2.
выраж1|выраж2	Строка, соответствующая либо выраж1, либо выраж2.
(…)	Используются для группировки выражений
\символ	Экранирует специальный символ, т.е. делает его обычным.

Например, регулярное выражение '^A([0-9]+|[^0-9])B?' означает: «В начале строки должна стоять буква A, за которой может следовать либо одна или несколько цифр, либо ровно один символ, отличный от цифры. После этого должна следовать буква B». То же самое условие можно записать и проще: '^A.[0-9]*B?'.

Регулярные выражения рекомендуется заключать в апострофы, чтобы шелл не попытался интерпретировать некоторые знаки как свои специальные символы.

Команда grep может иметь ряд ключей, некоторые из них приведены в следующей таблице.

-c	Выдается только число подходящих строк, а не сами строки.
-n	Перед каждой строкой выводится ее номер в файле.
-i	Игнорируется различие строчных и прописных букв.
-h	Отменяется выдача имен файлов.
-v	Выдаются только строки, которые НЕ содержат образца.
-r	Ищет во всех файлах указанного каталога и его подкаталогов.

Команда возвращает код завершения 0, если удалось хотя бы раз найти искомый образец (если задан ключ -v, то наоборот, если не удалось).

sort [ключи] [-k от_поля [, до_поля]] [список_файлов]

Команда сортирует строки указанных файлов или стандартного ввода и выводит результат (сортированный файл) на стандартный вывод. Если указано несколько файлов, они объединяются перед сортировкой.

Каждая строка рассматривается как набор полей, разделенных пробелами или символами табуляции. Параметр от_поля указывает, какое первое поле, считая от начала строки, следует учитывать при сравнении строк. Если параметр не задан, учитываются поля, начиная с первого. Параметр до_поля указывает, какое последнее поле должно участвовать в сравнении. Если параметр не задан, учитываются поля до конца строки.

Допускается указывать параметры от_поля и до_поля не в виде одного числа, а в виде пары чисел «m.n», где m – номер поля, а n – номер символа в поле. Как поля, так и символы в поле нумеруются начиная с 1.

Ключи команды определяют способ сортировки. По умолчанию строки сортируются по возрастанию, как в словаре. Некоторые ключи приведены в таблице.

-b	Игнорируются пробелы и табуляции в начале строки.
-f	Игнорируется различие между прописными и строчными буквами.
-n	Поля рассматриваются как числа (возможно, со знаком и десятичной точкой) и сравниваются по числовому значению.
-r	Сортировка ведется по убыванию.
-t символ	Указанный символ рассматривается как разделитель полей (вместо пробела и табуляции).
-o файл	Результат записывается в указанный файл (вместо стандартного вывода).
-u	Из нескольких строк с одинаковыми значениями сравниваемых полей сохраняется только одна.

cmp [-l |-s] файл1 файл2 [смещение1 [смещение2]]

Сравнивает данные в двух файлах. Если файлы идентичны, возвращает код завершения 0, если различаются – код 1, если произошла ошибка (например, файл не найден) – код, больший 1.

По умолчанию выдает номер байта и номер строки, в которых найдено первое различие. Нумерация начинается с 1. Если один файл совпадает с начальной частью другого, выдается сообщение о найденном конце файла. Если файлы идентичны, ничего не выдается.

Если задан ключ –l, то для каждого различия выдается номер байта и различающиеся значения. С ключом –s не выдается ничего (только код завершения).

Величины смещений указывают, сколько байт надо пропустить от начала каждого файла, прежде чем начать сравнение.

cut [ключи] [файлы]

Команда выделяет из указанных файлов или из стандартного ввода части строк, заданные либо номерами полей, либо номерами позиций в строке. Нумерация ведется с 1. Полем считается часть строки, ограниченная символами табуляции. Выделенные части строк выдаются на стандартный вывод.

Ключи команды приведены в таблице.

-c список_позиций	Задает номера выделяемых позиций. Можно использовать запятые и дефисы, например, список «10,20-30,40» означает «символы в позиции 10, в позициях с 20 до 30 и в позиции 40». Можно задавать неполные диапазоны, например, -25 (от начала до позиции 25) или 12- (с позиции 12 до конца строки). Этот ключ несовместим с остальными ключами.
-f список_полей	Задает номера выделяемых полей. Можно использовать запятые и дефисы.
-d символ	Указанный символ рассматривается как разделитель полей (вместо

	табуляции).
-s	Пропускаются все строки, не содержащие разделителя полей. Действует только при заданном ключе -f.

Создание текстового файла

Простейший способ создать небольшой файл – использовать команду echo с перенаправлением стандартного вывода:

```
$ echo -e "Hello!\nHow are you?" > hello
```

Другой вариант – использовать команду cat, опять-таки с перенаправлением вывода. Текст файла можно задать как стандартный ввод, содержащийся в тексте команды (в режиме «документ здесь»):

```
$ cat > hello << _TEXT_
> Hello!
> How are you?
> _TEXT_
```

Здесь в качестве ограничителя текста можно использовать любое слово, не встречающееся в этом тексте.
Еще один вариант – команда с перенаправлением стандартного вывода, но без перенаправления стандартного ввода. После ввода команды «cat > hello» шелл будет принимать вводимые строки текста, пока пользователь не введет комбинацию Ctrl+D (конец файла).
Здесь не рассматривается такой наиболее обычный способ создания файлов, как использование текстовых редакторов, входящих в состав UNIX.

Работа с файлами и каталогами

Чтобы выдать список всех имен файлов текущего каталога, можно с равным успехом использовать либо команду «**echo ***», либо команду ls без параметров. Имена будут выданы в одну строку и разделены пробелами. Если нужен иной (не текущий) каталог, можно указать его имя, например: «**echo home/student/***» или «**ls home/student**».

Команда «ls -l» выдает имена файлов по одному в строке, сопровождая имена основной информацией о файлах.

Если нужно выдать лишь имена файлов, соответствующие некоторому шаблону, то следует указать в команде требуемый шаблон, например: «**ls a*[123]**» – выдать все имена файлов, начинающиеся с буквы a и заканчивающиеся одной из цифр 1, 2 или 3.

Команда cat выдает уже не имена, а содержимое заданных файлов. Например, «*cat * > sumfile*» означает: «объединить содержимое всех файлов текущего каталога и записать результат в файл sumfile».

Рассмотрим в качестве объекта экспериментов файл **/etc/passwd**. Ниже приведено несколько строк из этого файла.

```
root:x:0:0:root:/root:/bin/bash
daemon:x:2:2:daemon:/sbin:
adm:x:3:4:adm:/var/adm:
operator:x:11:0:operator:/root:
```

```
games:x:12:100:games:/usr/games:
nobody:x:99:99:Nobody:/:
student:x:501:501:Student:/home/student:/bin/bash
```

Каждая запись состоит из 7 полей, разделенных двоеточиями. Чтобы, например, отсортировать файл по полным именам пользователей (поле 5), следует выполнить команду **«sort –k 5,5 –t ':' /etc/passwd».**

Чтобы выдать только строки, касающиеся пользователей, для которых при входе не запускается никакой шелл, можно использовать команду «**grep ':$' /etc/passwd**» (ищутся строки, в которых последним символом является двоеточие).

Чтобы выделить из списка только имена входа пользователей, можно воспользоваться командой **«cut –f 1 /etc/passwd»**.

Чтобы выдать данные о трех первых по алфавиту пользователях, можно использовать конвейер: **«sort /etc/passwd | head –n 3»**

Определить трех пользователей системы, чьи домашние каталоги с содержащимися в них файлами занимают больше всего дискового пространства

Все домашние каталоги имеют вид /home/имя_пользователя. Список всех домашних каталогов можно задать как /home/*. Получить объемы использованного дискового пространства позволяет команда du. В данном случае нас интересуют только суммарные значения, без разбивки по подкаталогам, поэтому следует задать ключ –s. Выполнив команду du в интерактивном режиме, можно обнаружить, что первый столбец выдаваемых данных содержит числа (количество занимаемых блоков), а второй столбец, отделенный знаком табуляции, содержит имена каталогов. Требуется отсортировать выдачу по убыванию числового значения в первом столбце. Для этого подойдет команда **«sort -rn».** Чтобы выбрать первые три строки в отсортированном списке, можно использовать команду **«head –n 3».**

Связывая все эти команды в один конвейер, можно записать:

du –s /home/* | sort –rn | head –n 3

Однако полученные в результате три строки будут содержать данные примерно такого вида:

```
120 /home/student
54 /home/dr
40 /home/user1
```

Чтобы отрезать ненужные начала строк, можно, например, подсчитать, что имена пользователей всегда содержатся во втором поле строки и начинаются с седьмой позиции этого поля. Можно дополнить конвейер двумя командами cut, первая из которых вырезает из строки второе поле, а вторая вырезает символы, начиная с седьмого:

du –s /home/* | sort –rn | head –n 3 | cut –f 2 | cut –c 7	Выдать имена всех файлов каталога-параметра и всех его подкаталогов, отступами показывая вложенность каталогов. Если параметр не задан, начать с текущего каталога.

Дадим скрипту имя lstree. Синтаксис вызова скрипта будет следующий:

```
lstree начальный_каталог
```

Для выдачи вложенной структуры каталогов удобнее всего использовать рекурсивный вызов скрипта *lstree*. Скрипт будет иметь два параметра: первый – начальный каталог, второй, предназначенный только «для служебного пользования» –

строка из нескольких точек, используемая для сдвига вправо при выдаче вложенных каталогов. Предполагается, что при первом вызове второй параметр задаваться не будет. Разумеется, это не очень корректное решение, зато самое простое. Более аккуратным решением было бы внутри скрипта определить рекурсивную функцию, но в данной работе функции шелла не рассматриваются.

Ниже приведен текст скрипта *lstree*.

```
#!/bin/bash
newdir=$1
indent=$2
[ -n $newdir ] && cd $newdir
for i in * ; do
 echo $indent$i;
 if [ -d $i ] && [ -x $newdir ] ; then
 lstree $i "$indent.."
 fi
done
```

Две переменные newdir и indent введены только для наглядности как копии позиционных параметров $1 и $2.

В начале работы скрипта, если задан непустой первый параметр, выполняется переход в задаваемый им каталог. Следует напомнить, что изменение текущего каталога будет действовать только на время выполнения скрипта. Далее в цикле выдаются имена всех файлов нового текущего каталога, сдвинутые вправо на строку, состоящую из точек indent. Если очередной файл является каталогом и для него имеется право на «выполнение» (для каталога это право позволяет сделать каталог текущим), то для него рекурсивно вызывается тот же скрипт lstree, ему передается имя каталога в качестве первого параметра и строка сдвига, расширенная на две точки, в качестве второго параметра.

Работа с учетными записями пользователей

По заданному имени входа пользователя выдать, в зависимости от указанного ключа, либо его полное имя, либо идентификатор пользователя, либо идентификатор группы пользователя. Допускается задание сразу нескольких ключей.

Пусть скрипт носит имя userinfo, а допустимые ключи – n (полное имя), u (идентификатор пользователя) и g (идентификатор группы). Синтаксис вызова:

```
userinfo [-nug] имя_входа
```

Ниже приведен текст скрипта.

```
#!/bin/bash
info=
while getopts nug option ; do
 case $option in
 n) info=$info" `grep '^'$2 /etc/passwd | cut –f 5 –d :`"
 u) info=$info" `grep '^'$2 /etc/passwd | cut –f 3 –d :`"
 g) info=$info" `grep '^'$2 /etc/passwd | cut –f 4 –d :`"
 *) echo Bad option: $option
 esac
done
echo $info
```

Требуемая информация выбирается из системного файла /etc/passwd с помощью команды grep, при этом искомой строкой является имя входа ($2). Символ ^ перед именем заставляет искать имя только в начале строки. Этим гарантируется невозможность ложного сравнения в случае, когда имя пользователя случайно совпадает с частью какой-либо совсем другой строки файла passwd.

Используется вызов конвейера в обратных апострофах, чтобы собрать стандартный вывод в переменной info. При вырезании нужного поля из строки файла passwd учитывается, что разделителем полей в этом файле является двоеточие.

Приведенное решение не идеально. В случае нескольких ключей команда grep будет вызываться несколько раз ради поиска одной и той же строки. Лучше было бы получить строку один раз, запомнить ее в переменной, а затем только вырезать из этой переменной нужные подстроки. Это можно сделать, либо используя конструкцию «документ здесь» в качестве стандартного ввода команды cut, либо используя для выделения подстроки из переменной команду expr, которая не рассматривалась в данном пособии.

Управление программами

Команды **true** и **false** служат для установления требуемого кода завершения процесса: true - успешное завершение, код завершения 0; false - неуспешное завершение, код может иметь несколько значений, с помощью которых определяется причина неуспешного завершения. Коды завершения команд используются для принятия решения о дальнейших действиях в операторах цикла **while** и **until** и в условном операторе **if**. Многие команды LINUX вырабатывают код завершения только для поддержки этих операторов.

Условный оператор if проверяет значение выражения. Если оно равно true, Shell выполняет следующий за **if** оператор, если false, то следующий оператор пропускается. Формат оператора **if**:

if <условие>
then
list1
else
list2
fi

Команда **test** (проверить) используется с условным оператором **if** и операторами циклов. Действия при этом зависят от кода возврата **test**. **Test** проводит анализ файлов, числовых значений, цепочек символов. Нулевой код выдается, если при проверке результат положителен, ненулевой код при отрицательном результате проверки.

В случае анализа файлов синтаксис команды следующий:

test [-rwfds] file

где

-r – файл существует и его можно прочитать (код завершения 0);
-w – файл существует и в него можно записывать;
-f – файл существует и не является каталогом;
-d – файл существует и является каталогом;
-s – размер файла отличен от нуля.

При анализе числовых значений команда **test** проверяет, истинно ли данное отношение, например, равны ли A и B . Сравнение выполняется в формате:

-eq	A = B
-ne	A <> B
test A -ge B	A >= B
-le	A <= B

-gt A > B
-lt A < B

Отношения слева используются для числовых данных, справа – для символов.

Кроме команды **test** имеются еще некоторые средства для проверки:

! - операция отрицания инвертирует значение выражения, например, выражение **if test true** эквивалентно выражению **if test ! false**;

o - двуместная операция "ИЛИ" (or) дает значение true, если один из операндов имеет значение true;

a - двуместная операция "И" (and) дает значение true, если оба операнда имеют значение true.

Циклы

Оператор цикла с условием **while** true и **while** false. Команда **while** (пока) формирует циклы, которые выполняются до тех пор, пока команда **while** определяет значение следующего за ним выражения как true или false. Формат оператора цикла с условием **while** true:

while list1
do
list2
done

Здесь list1 и list2 - списки команд. **While** проверяет код возврата списка команд, стоящих после **while**, и если его значение равно 0, то выполняются команды, стоящие между **do** и **done**. Оператор цикла с условием **while** false имеет формат:

until list1
do
list2
done

В отличие от предыдущего случая условием выполнения команд между **do** и **done** является ненулевое значение возврата. Программный цикл может быть размещен внутри другого цикла (вложенный цикл). Оператор **break** прерывает ближайший к нему цикл. Если в программу ввести оператор **break** с уровнем 2 (**break 2**), то это обеспечит выход за пределы двух циклов и завершение программы.

Оператор **continue** передает управление ближайшему в цикле оператору **while**.

Оператор цикла с перечислением **for**:

for name in [wordlist]
do
list
done

где name - переменная; wordlist - последовательность слов; list - список команд. Переменная name получает значение первого слова последовательности wordlist, после этого выполняется список команд, стоящий между **do** и **done**. Затем name получает значение второго слова wordlist и снова выполняется список list. Выполнение прекращается после того, как кончится список wordlist.

Ветвление по многим направлениям **case**. Команда **case** обеспечивает ветвление по многим направлениям в зависимости от значений аргументов команды. Формат:

case <string> in
s1) <list1>;;
s2) <list2>;;
.
.
.

sn) <listn>;;
***) <list>**
esac

3десь list1, list2 ... listn - список команд. Производится сравнение шаблона string с шаблонами s1, s2 ... sk ... sn. При совпадении выполняется список команд, стоящий между текущим шаблоном sk и соответствующими знаками ;;. Пример:

echo -n 'Please, write down your age'
read age
case $age in
test $age -le 20) echo 'you are so young' ;;
test $age -le 40) echo 'you are still young' ;;
test $age -le 70) echo 'you are too young' ;;
***)echo 'Please, write down once more'**
esac

В конце текста помещена звездочка * на случай неправильного ввода числа.

Конвейеризация и перенаправление в ОС LINUX.

Средство, объединяющее стандартный выход одной команды со стандартным входом другой команды называется *конвейером* и обозначается " | "

Пример: cat f1 | grep –h result | sort | cat –b>f2

Данный конвейер из файла f1 (cat) выберет все строки, содержащее слово result (grep) , отсортирует (sort) полученные строки, а затем пронумерует (cat – b) i выведет результат в файл f2)_

Поскольку устройства в ОС LINUX представлены специальными файлами, их можно использовать при перенаправлениях. Специальные файлы находятся в каталоге /dev.

Пример, lp – печать, console – консоль, ttyi – и-ый терминал, nul - - фиктивный (пустой файл)

ls>/dev/lp - выведет содержимое текущего каталога на печать
f1</dev/null - обнулит файл f1
sort f1 | tee / dev / lp | tail – 20 - будет отсортирован файл и передан на печать , а 20 последних строк также будут выданы на экран
wc -1<f1 – подсчитает и выведет на экран число строк в файле f1
wc -1<f3>f4 и wc -1>f4<f3 выполняются одинаково: подсчитываются число строк файла f3 и результат помещается в файл f4

Группировка команд:

; и – определяют последовательность выполнения команд

& - фоновое выполнение команд(при выполнении команды в фоновом режиме на экран выводится номер процесса, соответствующий выполняемой команде и система, запустив этот процесс, вновь выходит на диалог с пользователем)

&& - выполнение последующей команды при условии нормального завершения предыдущей, иначе игнорировать

|| - выполнение последующей команды при ненормальном завершении предыдущей, иначе игнорировать

Фоновые процессы можно уничтожить. Для этого необходимо знать его номер. Если этот номер забыт или надо убедиться, что он не закончен служит команда: **ps –aux**

В выведенной таблице можно найти номер процесса, подлежащий уничтожению и тогда командой

Kill -9 866 849

можно уничтожить эти процессы.

Порядок выполнения работы

Задание1. Создайте скрипт, выполняющий следующую последовательность действий:

1. Создать в домашнем каталоге личный каталог
2. Создать в личном каталоге текстовый файл pr.txt, который содержит справку о команде cat
3. Добавить в файл pr.txt полное имя текущего каталога
4. Добавить в текстовый файл pr.txt содержимое текущего каталога в длинном формате (т.е. с информацией о размере каждого файла в блоках)
5. Поместить в файл pr2.txt все строки из pr.txt, которые содержат слово «команда»
6. Пронумеровать все строки файла pr2.txt
7. Вывести на экран содержимое файла pr2.txt
8. Вывести на экран число строк в файле pr2.txt
9. Написать конвейер, который выведет на экран число строк текущего каталога
10. Запустить фоновый процесс, который ищет файл с именем “conf”
11. Уничтожить этот процесс

Задание 2. Составьте и выполните shell – программу по вариантам, включающей следующие действия:

1. Для каждого из файлов, перечисленных в списке параметров, создать отдельный подкаталог своего домашнего каталога и переместить туда файл. В случае, если нельзя выполнить перемещение (нельзя удалить файл), запрашивать пользователя, выполнять ли копирование или пропустить файл. Имена подкаталогов строить путем добавления к имени домашнего каталога чисел 1, 2, 3 и т.д.
2. Создать вручную «телефонный справочник», состоящий из нескольких записей, содержащих 3 поля: фамилия, адрес, номер телефона. Поля записи разделять знаком табуляции. Составить скрипт, который по заданной фамилии или адресу, или номеру телефона (в зависимости от указанного ключа) выдает значения двух других полей соответствующей записи.
3. Провести копирование из одного каталога (источника) в другой каталог (приемник) всех файлов, имена которых удовлетворяют заданному шаблону. В зависимости от заданного ключа, запрашивать подтверждение копирования либо для каждого файла, либо только в случае замены существующего файла, либо никогда.
4. Выполнить в диалоге настройку поиска файла: запросить и ввести шаблон имени, начальный каталог поиска, тип файла, число дней после изменения файла или после обращения к нему. Выполнить поиск и вывести имена найденных файлов.
5. Найти в указанном каталоге все файлы, содержащие заданную строку. Для каждого найденного файла запросить действие, которое необходимо выполнить: удалить файл, запретить доступ к нему прочих пользователей или оставить, как есть.
6. Вывод на экран списка параметров командной строки с указанием номера каждого параметра.
7. Присвоение переменным A, B и C значений 10, 100 и 200, вычисление и вывод результатов по формуле D=(A*2 + B/3)*C.
8. Формирование файла со списком файлов в домашнем каталоге, вывод на экран этого списка в алфавитном порядке и общего количества файлов.
9. Переход в другой каталог, формирование файла с листингом каталога и возвращение в исходный каталог.
10. Запрос и ввод имени пользователя, сравнение с текущим логическим именем пользователя и вывод сообщения: верно/неверно
11. Запрос и ввод имени файла в текущем каталоге и вывод сообщения о типе файла.
12. Циклическое чтение системного времени и очистка экрана в заданный момент.
13. Циклический просмотр списка файлов и выдача сообщения при появлении заданного имени в списке.

Контрольные вопросы

1. Какое назначение имеют shell - файлы?
2. Как создать shell - файл и сделать его выполняемым?
3. Какие типы переменных используются в shell - файлах?
4. В чем заключается анализ цепочки символов?
5. Какие встроенные команды используются в shell - файлах?
6. Как производится управление программами?
7. Назовите операторы создания циклов.
8. Как задаются и выполняются простые и сложные команды?
9. Какие функции выполняет командный интерпретатор *Shell*?