ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА
(РОСАВИАЦИЯ)

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)

Кафедра вычислительных машин, комплексов, систем и сетей

Н.И. Романчева

ИНФОРМАТИКА

ОС LINUX: ОСНОВЫ РАБОТЫ И ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ КОМАНДНОГО ИНТЕРПРЕТАТОРА

Учебно-методическое пособие
по выполнению лабораторных работ № 1–3

для студентов
направления 25.03.01 и специальности 10.05.02
очной формы обучения

Москва
ИД Академии Жуковского
2024

УДК 004.451
ББК 6Ф7.3
Р69

Рецензент:
Терентьев А.И. – канд. техн. наук

Романчева Н.И.

Р69 Информатика. ОС Linux: основы работы и программирование на языке командного интерпретатора [Текст] : учебно-методическое пособие по выполнению лабораторных работ № 1–3 / Романчева Н.И. – М.: ИД Академии Жуковского, 2024. – 48 с.

Данное учебно-методическое пособие издается в соответствии с учебным планом для обучающихся по направлению подготовки 25.03.01 (направленность (профиль) «Интеллектуальные системы сопровождения технической эксплуатации АТ») и специальности 10.05.02 «Информационная безопасность телекоммуникационных систем» (специалитет) очной формы обучения.

Рассмотрено и одобрено на заседаниях кафедры 23.01.2024 г. и методических советов по направлению подготовки 25.03.01 –23.01.2024 г., по специальности 10.05.02 – 23.01.2024 г.

УДК 004.451
ББК 6Ф7.3

В авторской редакции

Подписано в печать 14.06.2024 г.
Формат 60х84/16 Печ. л. 3 Усл. печ. л. 2,79
Заказ № 1015/0410-УМП03 Тираж 30 экз.

Московский государственный технический университет ГА
125993, Москва, Кронштадтский бульвар, д. 20

Издательский дом Академии имени Н. Е. Жуковского
125167, Москва, 8-го Марта 4-я ул., д. 6А
Тел.: (499) 755-55-43
E-mail: zakaz@itsbook.ru

С О Д Е Р Ж А Н И Е

1. ОСНОВНЫЕ ТРЕБОВАНИЯ И ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

Настоящее учебно-методическое пособие предназначено для студентов по направлению подготовки 25.03.01 (направленность (профиль) «Интеллектуальные системы сопровождения технической эксплуатации АТ» и специальности 10.05.02 Информационная безопасность телекоммуникационных систем (специалитет), выполняющих лабораторные работы по дисциплине Информатика и информационные технологии в соответствии с ФГОС3++. В данное учебно-методическое пособие включены материалы для выполнения лабораторных работ № 1-3.

Продолжительность каждой лабораторной работы - 4 часа.

Целью проведения лабораторных работ является закрепление основных теоретических положений, изложенных в лекциях по дисциплине Информатика.

В процессе выполнения лабораторных работ осуществляется:

- закрепление основных теоретических положений, изложенных в лекциях на примере широко используемых в различных областях ОС UNIX/Linux (Red Hat и др.);
- освоение приемов и методов работы с текстовыми и графической консолями;
- получение навыков настройки прав доступа к файлам;
- освоение приемов работы с встроенным редактором программ Midnight Commander;
- использование технологии программирования на языке Bourn shell.
- получение навыков и умений программирования на языке командного интерпретатора, а также навыки отладки программ на языке С для ОС с открытым кодом.

Лабораторная работа состоит из следующих этапов:

1) домашняя подготовка;

2) выполнение работы на компьютере в соответствии с заданием;

3) сдача выполненной работы преподавателю на персональном компьютере;

4) распечатка результатов работы на принтере;

5) оформление отчета;

6) защита лабораторной работы.

В процессе домашней подготовки студент:

- изучает лекционный материал,
- материал по теме лабораторной работы данного учебно-методического пособия и дополнительной литературы;
- знакомится с заданием на выполнение лабораторной работы;
- готовит проект отчета по выполнению лабораторной работы.

ВЫПОЛНЕНИЕ лабораторной работы проводится во время занятий в лаборатории кафедры ВМКСС МГТУГА в присутствии преподавателя в соответствии с расписанием, утвержденным проректором по УМР МГТУ ГА. В процессе выполнения лабораторной работы студент последовательно выполняет задание. По завершению работы - демонстрирует преподавателю результаты.

СДАЧА РАБОТЫ преподавателю на персональном компьютере заключается в демонстрации выполненной работы и выполнении непосредственно при преподавателе индивидуального дополнительного задания.

ОТЧЕТ по каждой лабораторной работе должен содержать: название работы; цель лабораторной работы; задание на выполнение лабораторной работы; краткие комментарии по выполнению лабораторной работы; распечатки файлов результатов или диаграмм, подписанные преподавателем.

ЗАЩИТА лабораторной работы преподавателю проводится по контрольным вопросам и при наличии оформленного отчета (распечатки результатов выполнения лабораторной работы должны быть аккуратно приклеены). После защиты лабораторной работы преподавателем делается соответствующая запись на отчете студента.

РЕЙТИНГОВЫЙ КОНТРОЛЬ по лабораторным работам производится при их сдаче во время лабораторных занятий. Перед выполнением лабораторной работы производится экспресс-опрос для определения готовности студентов к выполнению работы (знания теоретического материала, целей работы и т.д.).

1) Допуск к ЛР

Допуск к выполнению ЛР происходит в виде устного опроса (при условии наличия у студента печатной версии титульного листа отчета по лабораторной работе).

Студент, не прошедший устный опрос к выполнению лабораторной работы не допускается.

2) Выполнение и защита ЛР

Минимальная оценка выставляется за выполненную и защищенную лабораторную работу, а дополнительными баллами оценивается качество выполненной лабораторной работы и полнота знаний, показанная студентами при ее защите. Лабораторная работа считается сделанной, если она выполнена полностью в соответствии с заданием.

Сроки сдачи лабораторных работ

Номер лабораторной работы	Лабораторная работа	Срок сдачи*
1	Лабораторная работа №1	Лабораторная работа №1
2	Лабораторная работа №2	Лабораторная работа №2
3	Лабораторная работа №3	Лабораторная работа №3

**Срок сдачи лабораторной работы – в соответствии с расписанием проведения лабораторных работ.*

3) Отчет по ЛР

Отчет по лабораторной работе представляется в печатном виде в формате, предусмотренном шаблоном отчета по лабораторной работе.

4) Защита ЛР

Отчет не может быть принят и подлежит доработке в случае:

- некорректной обработки результатов выполнения лабораторной работы;
- небрежного выполнения;
- низкого качества представленного материала (неполное решение);
- отсутствия необходимых разделов отчета;
- отсутствия необходимого графического материала;
- отсутствия выводов по работе и т.п.

2. ЛАБОРАТОРНАЯ РАБОТА №1
ОС LINUX: ОСНОВЫ РАБОТЫ, ИССЛЕДОВАНИЕ СИСТЕМЫ

2.1. Цель работы

Целью данной работы является получение практических навыков и умений работы в ОС Linux (Red Hat и др.), а также исследование системы.

2.2. Перечень компетенций, формируемых в ходе выполнения лабораторной работы

Универсальная компетенция УК-2:

Способен осуществлять критический анализ проблемных ситуаций на основе системного подхода, вырабатывать стратегию действий:

- умеет работать на современных компьютерах в части конфигурирования и монтирования файловых систем;

Общепрофессиональная компетенция ОПК-2:

Способен применять информационно-коммуникационные технологии, программные средства системного и прикладного назначений, в том числе отечественного производства, для решений задач профессиональной деятельности.

- умеет: монтировать файловые системы; работать с терминальными консолями;

2.3. Задание на выполнение работы

1) Загрузить операционную систему. Основные этапы монтирования системы внести в отчет по лабораторной работе.
2) Идентифицировать себя с помощью одной из регистрационных записей обычного пользователя (*login – user, pass- user123*).
3) Используя команду pwd (**P**rint **W**orking **D**irector), определить местоположение пользователя в системе.
4) Определить количество текстовых и графических консолей. Выполнить переключение между консолями (используя горячие клавиши).
5) Создать текстовый файл, используя команду cat, touch, nano.
6) Дополнить текстовый файл (cat >>1.txt) и вывести на экран.
7) Установите права на файл: чтение файла только у пользователя, являющегося владельцем файла; запись только у группы, являющейся владельцем файла, установить права на исполнение (chmod) и т.д.
8) Просмотреть файлы /etc/passwd , /etc/shadow (под root).
9) Просмотреть файл /etc/group.
10) Определить, находится ли пользователь user2 в системе.
11) Изменить владельца текущего сеанса и просмотреть информацию о процессе (использовать команды su и ps).
12) Получить список файлов текущего каталога с указанием размера, времени создания и изменения, имени владельца, таблицы прав.

13) Вывести постранично список скрытых файлов каталога /etc (ls –a|less)
14) Используя текстовые консоли выполнить:
- ввод и вывод основных команд;
- перенаправление команд;
- использование каналов;
- настройку прав доступа;

15) Выполнить поочередно поиск файлов: /etc/passwd, file.txt, а также группу файлов по шаблонам: *.txt, *.exe, L*, L??*, [ab]*a.
16) Найти в текущем каталоге все файлы, в именах которых встречается хотя бы один символ в верхнем регистре.
17) Изучить интерфейс и выполнить основные файловые операции, используя встроенный редактор программ Midnight Commander (mc).
18) Отобразить в отчете дерево корневой файловой системы.
19) Провести исследование системы:

а) получить информацию о пользователях, находящихся в системе;

б) получить информацию о пользователях и процессах, недавно завершивших работу.

в) определить, кто и когда зарегистрировался в системе.

г) получить информацию о зарегистрированных в настоящий момент пользователях.

д) определить, сколько свободного дискового пространства и индексных дескрипторов доступно в разделе смонтированного диска.

е) оценить стабильность работы и загрузку системы.

ж) получить информацию о таблице взаимодействия процессов.

19) Запустить графическую оболочку (startx) и выполнить задания пункта 13.
20) Запустить в графической оболочке терминальную консоль и перейти в режим работы суперпользователя.
21) Просмотреть файлы /etc/shadow.
22) Просмотреть файл fstab, указав к нему полный путь.
23) Выполнить монтирование сменных устройств (flach) средствами графической оболочки.
24) Проверить командой df, что сменное устройство размонтировано.
25) Скопировать на сменное устройство созданный файл (п.5 задания).
26) Размонтировать сменное устройство.
27) Добавить пользователя N в системе, используя команду adduser (useradd).Имя пользователя N должно содержать не менее 6 символов и совпадать с паролем.
28) Результаты исследований занести в отчет.

2.4. Основные теоретические сведения и приемы работы

2.4.1. Назначение ОС Linux

ОС Linux - название ядра операционной системы, базового программного обеспечения низкого уровня, которое управляет аппаратной частью

компьютера. В то же время это полноценная операционная система включает набор утилит и прикладных программ, который может варьироваться. Все компоненты системы, включая исходные тексты, распространяются с лицензией на свободное копирование и установку для неограниченного числа пользователей. Она поддерживает стандарты открытых систем и протоколы сети Internet.

В зависимости от состава программных компонент выделяют следующие дистрибутивы Linux: Red Hat, Caldera, SuSe, Debian, Slackware и т.д.

При выполнении лабораторной работы используется Red Hat Linux - многозадачная, многопользовательская сетевая операционная система. Многозадачность - очень важное достоинство Linux. Система устроена так, что под каждую задачу, выполняемую пользователем, выделяется определенное количество ресурсов. Ресурсы компьютера, такие как, например, оперативная память, не передаются приоритетной задаче (как это делается в Windows), а используются параллельно несколькими приложениями. Это повышает производительность системы и снижает риск ее «зависания».

2.4.2. Вход в систему и идентификация пользователя

После полной загрузки операционной системы появится подсказка, имеющая примерно следующий вид:

```
Red Hat Linux release 6.0 (Hedwig)
Kernel 2.2.5-22 on an i686
login:
```

Далее следует ввести имя пользователя и пароль. По имени пользователя система опознает (идентифицирует) Вас как одного из пользователей, которые могут работать в системе. Для идентификации пользователя:

- введите в поле ввода *Login:* регистрационное имя обычного пользователя.

После ввода имени пользователя система выполняет аутентификацию, для чего требуется ввести пароль, соответствующий данному пользователю.

- введите в поле *Password:* пароль пользователя.

Если пароль или имя пользователя введены неправильно, система потребует снова ввести имя пользователя и пароль. При корректном имени и пароле запускается одна из консолей или графическая оболочка .

Если при установке системы не было создано ни одной регистрационной записи обычного пользователя, введите имя суперпользователя *root* и добавьте соответствующую запись.

2.4.3. Добавление регистрационной записи обычного пользователя

Для того чтобы добавить регистрационную запись обычного пользователя, необходимо:

- перейти в текстовую консоль или режим эмуляции терминала и войти в систему как суперпользователь;

- в поле ввода *Имя пользователя* ввести имя пользователя, состоящее из маленьких латинских букв и цифр и начинающееся с буквы;

- в поле ввода *Полное имя* ввести обычное имя пользователя, записанное латинскими буквами (например, Ivan Ivanov);

- в поле ввода *Пароль* ввести пароль этого пользователя;

- в поле ввода *Подтвердите пароль* ввести этот же пароль повторно, далее нажать на кнопку *Добавить*.

Команда добавления пользователя *adduser* создает регистрационную запись для нового пользователя (для чего вносит изменения в ряд конфигурационных файлов системы) и создает так называемый домашний каталог */home/имя_пользователя*, содержащий некоторые необходимые файлы и подкаталоги. Этот каталог и находящиеся в нем файлы принадлежат пользователю, и он имеет полный набор прав для работы как с существующими, так и вновь создаваемыми объектами в этом каталоге.

Рекомендуется создать хотя бы одного обычного пользователя. Не рекомендуется пользоваться правами суперпользователя без необходимости.

2.4.4. Удаление регистрационной записи обычного пользователя

Чтобы удалить ошибочно созданную регистрационную запись:

- необходимо работать с правами суперпользователя;
- ввести в командной строке команду *userdel имя _пользователя*;

Для удаления регистрационной записи, а также принадлежащих пользователю файлов, необходимо ввести в командной строке команду

userdel - r имя _пользователя

Эта команда удаляет каталог */home/имя_пользователя* и все его содержимое.

Удаление регистрационной записи пользователя невозможно, если он в данный момент зарегистрирован в системе или работает какой-либо процесс, запущенный от его имени.

2.4.5. Текстовые и графические режимы работы

В ходе загрузки системы создаются несколько консолей - виртуальных устройств ввода-вывода, которыми могут пользоваться различные компоненты и пользовательские программы системы. В стандартной настройке ОС работает с 7 виртуальными консолями (6 текстовых и 1 графическая), их которых в каждый момент времени только одна может быть связана с реальной (физической) консолью, т.е. является активной. Консоли, представляющие информацию только в текстовом виде с использованием экранных шрифтов в форматах видеосистемы компьютера, называются *текстовыми*. В таких консолях используется интерфейс командной строки. Другие консоли (*графические*) представляют информацию в графическом виде, используя Графический пользовательский интерфейс (GUI). Как правило, в одной из консолей автоматически запускается графическая среда X Windows System и графическая оболочка GNOME (рис.1). Для запуска графической оболочки необходимо ввести команду *startx*.

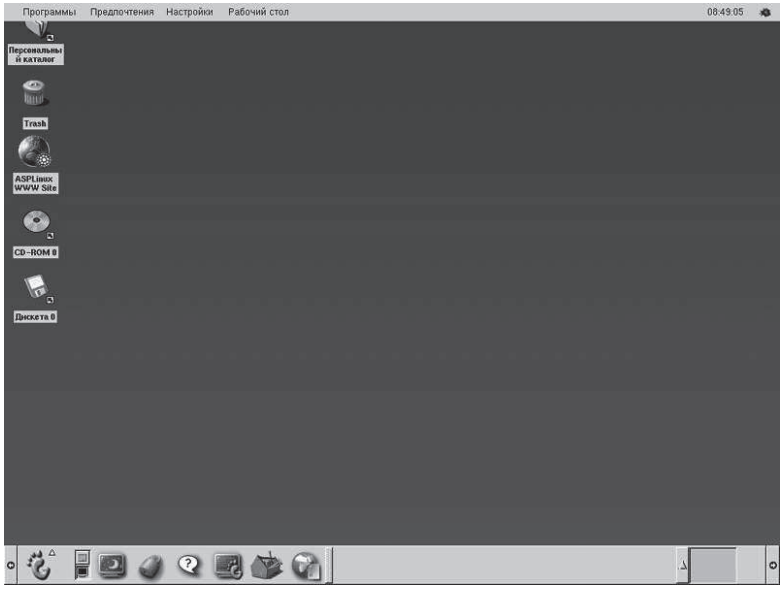

Рисунок 1. – Окно графической консоли

Для перехода между консолями используется сочетание клавиш [CTRL]+ [ALT]+ [Fn], n - номер консоли, находится в интервале от 1 до 12. Например, чтобы сделать активной консоль с номером 4, следует нажать клавиши [CTRL]+ [ALT]+ [F4].

2.4.6. Интерпретатор shell. Файлы и права к ним

В Linux у каждого пользователя обязательно есть свой домашний каталог, предназначенный для хранения всех собственных данных пользователя. Именно с этого каталога пользователь начинает работу после регистрации в системе. Домашние каталоги пользователей обычно собраны в каталоге /home, их название чаще всего совпадает с учётным именем пользователя в системе, например, для пользователя user домашним каталогом будет /home/user.

Пользователь является полным хозяином внутри своего каталога, однако остальная часть файловой системы доступна ему только для чтения, но не для записи. Доступ других пользователей к чужому домашнему каталогу ограничен, например, пользователи могут читать содержимое файлов друг друга, но не имеют права их изменять или удалять.

Файл принадлежит создавшему его пользователю, а также группе, членом которой данный пользователь является. Пользователи, имеющие доступ к файлу, делятся на три категории:

- Владелец файла, создавший его;
- Члены группы, являющейся владельцем файла;
- Остальные пользователи.

Владелец файла может самостоятельно определять, кому позволено производить запись в файл, читать его содержимое, а также запускать файл на выполнение, если он является исполняемым. Пользователь с правами root может отменить практически все ограничения, заданные пользователем.

Доступ к созданному файлу может осуществляться тремя способами:

- путем чтения, при этом содержимое файла отображается на экране;
- путем записи, при этом содержимое файла редактируется или удаляется;
- путем выполнения, если файл содержит сценарий интерпретатора shell либо является программой.

После создания файла система сохраняет о нем всю информацию, в частности:

- раздел диска, где физически находится файл;
- размер файла;
- идентификатор владельца файла, а также тех, кому разрешен доступ к файлу;
- индексный дескриптор;
- дата и время последнего изменения файла;
- режим доступа к файлу.

Изменять режим доступа к файлам можно, с помощью команды chmod. Аргументы этой команды могут быть заданы либо в числовом виде (абсолютный режим), либо в символьном (символьный режим).

Общий формат команды chmod для символьного режима:

chmod *[кто] оператор [разрешения] файл (список файлов))*

Значения параметра *кто:*

u Владелец файла,
g Группа, являющаяся владельцем файла,
o остальные пользователи,
a Все (владелец, группа и остальные пользователи)

Значения параметра *оператор:*

\+ Добавление разрешения,
\- Удаление разрешения,
= Установка заданного разрешения

Значения параметра *разрешения:*

r Право чтения,
w Право записи,
x Право выполнения,
X Установка права выполнения только в том случае, если для какой-либо категории пользователей уже задано право выполнения,
s Установка бита SUID или SGID для владельца или группы,
t Установка sticky-бита,
u Установка тех же прав, что и у владельца файла,
g Установка тех же прав, что и у группы, являющейся владельцем файла,

o Установка тех же прав, что и у других пользователей.

2.4.7. Обзор основных команд

Синтаксис многих команд предусматривает использование параметров довольно сложного формата, указываемых в командной строке после имени команды. Полное описание команд и их параметров можно получить, набрав

man имя_команды

Дополнительную информацию по команде:

info имя_команды

После ввода вышеперечисленных команд ОС Linux открывает на нескольких, сменяющих друг друга экранах описание нужной команды. Если не помните правильный синтаксис имени нужной команды, введите команду man с параметром -k, затем ключевое слово для поиска нужной команды. Система выполнит поиск в своих файлах справки, содержащей это ключевое слово. Для этой команды имеется также псевдоним apropos. Например, если ввести команду:

man ls

ОС Linux выведет на экран справку о команде ls, в том числе обо всех ее параметрах. По команде:

man -k cls

выводится список всех команд, в справке, о которых есть слово cls.

Команда apropos cls аналогична команде man -k cls.

Ниже приведены наиболее употребительные команды и наиболее частые форматы.

Команды перезагрузки и останова системы

Перезагрузить компьютер можно с помощью команды *reboot*. Для прекращения работы Linux также используются команды *halt, fasthalt fastboot*. Все названные команды представляют собой короткий вариант команды *shutdown* с определенными параметрами:

halt - shutdown -h now

fasthalt - shutdown -fh now

fastboot - shutdown -fr now

reboot - shutdown -r now

Параметры команды *shutdown* означают следующее:

-f - создать файл /fastboot и при следующей загрузке компьютера пропустить тестирование файловой системы;

-h - остановить систему;

-r - перезапустить систему.

Команды для работы с каталогами

В Linux есть множество команд для работы с каталогами. Как и в других операционных системах, каталоги в Linux можно удалять, создавать, перемещать, а также выводить информацию об их состоянии. В Linux, как и в DOS, файлы хранятся в каталогах, организованных в древовидные структуры. Файл можно указывать в виде пути из корневого каталога, обозначаемого

символом /, до файла. Таким образом, полное имя файла new, принадлежащего пользователю sanja, может иметь вид /home/sanja/new. В Linux есть понятие рабочего каталога пользователя. Рабочий каталог обычно обозначается символом ~ (тильда). Например, команда копирования файла из текущего каталога в рабочий может иметь вид *cp new ~*

Смена текущего каталога (cd)

Для перемещения по дереву каталогов Linux применяется команда cd. Для перехода в рабочий каталог эта команда вводится без параметров. Для перехода из одного каталога в другой формат команды тот же, что и DOS:
cd new-directory, где *new-directory* – имя каталога, в который следует перейти. Кроме того, в Linux текущий каталог представляется одной точкой (.), каталог-родитель - двумя (. .) - и, конечно же, в этом DOS наследует UNIX и Linux, а не наоборот.

Следует обратить внимание на символ разделителя каталогов. В DOS для этого применяется обратная косая черта (\), которая в Linux служит указателем продолжения команды с новой строки. В Linux каталоги разделяются прямой косой чертой (/). Кроме того, в DOS не имеет значения, отделены ли параметры (.) и (. .) пробелами от имени команды, в то время как в Linux это важно. Например, команда *cd..* не будет распознана командным интерпретатором, правильный формат которой – cd .. В Linux между командой и параметром обязательно должен быть пробел.

Команды управления файлами

ls [opt] [file1 file2 …] - вывод имен файлов текущего каталога. В качестве параметров можно задать имена каталогов, содержимое которых нужно вывести, или имена файлов, информацию о которых нужно получить. Опции команды позволяют получить список дополнительной информации:

ls - список файлов текущего каталога (краткий формат).

ls -al - получить список файлов текущего каталога с указанием размера, времени создания и изменения, имени владельца, таблицы прав и других данных, например:

-rwxr- - r - - 2 user group 34 Nov 10 10:34 a.out

где *-rwxr--r--* - права доступа (за исключением первого символа, обозначающего тип файла: - – обычный файл, d – каталог, p – именованный канал, b – специальное блочное устройство, c – специальное символьное устройство) на чтение (read -символ r), запись (write -символ w), выполнение (execute - символ x). Наличие прав обозначается соответствующим символом, а отсутствие - символом "-";

2 – число жестких связей (hard link) данного файла;

user - имя владельца - пользователя (user owner) файла. Владельцем-пользователем вновь созданного файла является пользователь, запустивший процесс, который и создал файл;

group – имя владельца - группы (group owner). Порядок назначения владельца группы зависит от конкретной версии UNIX;

34 - размер файла;

Nov 10 - дата последнего изменения;

10:34 - время последнего изменения;

a.out - имя файла.

ls –aC – просмотр скрытых файлов**;**

ls -C *user* **-** вывод списка файлов каталога *user* в несколько колонок в алфавитном порядке;

ls –RC */home/user/bin* **-** рекурсивный просмотр каталогов, например, */home/user/bin*;

ls –tC - сортировка по времени модификации, все вновь созданные файлы размещаются в начале списка;

ls –ctC - сортировка по изменению статуса (изменение владельца или прав доступа). Если ключ **t** не задан, то ключ **c** игнорируется.

cd [dir] - сменить текущий каталог. При задании без параметра – происходит переход в домашний каталог пользователя.

Команды работы с файлами

В Linux каталоги являются одним из типов файлов, поэтому для работы с теми и другими применяются одни и те же команды.

cp файл1 файл2 - копировать файл. Она применяется для копирования одного или нескольких файлов из одного каталога в другой. Если вместо имени второго файла указать каталог, то **файл1** копируется в каталог **файл2** с тем же именем, при этом в имени первого файла допускается использование подстановочного символа “звездочка”. Чтобы скопировать файл с тем же именем в качестве второго параметра, ставится точка (.). Команда *cp* аналогична команде *COPY* DOS.

rm файл1 – удалить файлы с указанными именами. Допускается использование подстановочного символа “звездочка” и другие специальные возможности. Например, команда *rm *m** позволит удалить все файлы, в именах которых встречается буква *m;*

mkdir [имя_каталога1]… – создать новый каталог.

В качестве параметра указывается имя создаваемого каталога, например: *mkdir /home/new* . Поскольку структура каталогов составляет основу файловой системы, в Linux имеется также команда создания каталога *mkdir*. В отличие от DOS, где можно воспользоваться псевдонимом данной команды md, в Linux надо вводить ее полное имя.

rmdir [имя_каталога]… – удалить пустой каталог. Удаление файлов в ОС Linux осуществляется при помощи команды *rm,* где *опции* - разнообразные параметры, позволяющие управлять процессом удаления; *файлы* - объекты файловой системы, которые необходимо удалить. Чтобы удалить файл при помощи команды rm, пользователь должен иметь разрешение на каталог, содержащий данный файл, однако ему не обязательно иметь разрешение для самого файла, который он хочет удалить. Наиболее часто используемыми, при работе с командой *rm*, являются следующие опции:

-f (force) - используется для принудительного удаления файлов без вывода запросов на подтверждение пользователя.;

-i (interactive) - используется для подачи запроса подтверждения пользователю перед удалением файлов;

-r (recursive) - используется для рекурсивного удаления содержимого каталога и самого каталога.

Работать с командой rm надо предельно внимательно, поскольку при работе в командном режиме не существует понятия «корзины» и файлы, удаленные данной командой будет возможно восстановить только из резервных копий (если таковые были своевременно сделаны) или используя низкоуровневые утилиты восстановления файловой системы, такие как debugfs.

ln [-опция] source target - создает жесткую связь имени *source* с файлом, адресуемым именем *target*. При использовании опции *–s* будет создана символическая ссылка;

pwd - вывести имя текущего каталога;

cmp [-опция] файл1 файл2 - сравнить два файла, указанных в качестве аргумента. Если файлы одинаковы, то никакое сообщение не выводится, в противном случае выводятся данные о первом несоответствии между этими файлами, например:

file1 file2 differ: char 15, line 6 ,

найдено различие в 15 символе 6-ой строки.

mv from-filename to-filename, где from-filename -перемещение файлов исходный файл; to-filename - новый файл. По команде mv, аналогичной команде MOVE из DOS, файлы перемещаются из одного каталога в другой. Действие этой команды аналогично действию команды копирования с последующим удалением исходных файлов. Команда mv не создает копий файлов.

Управление выводом на экран

cat [-опция] файл - выводит содержимое *файла* на экран терминала. В качестве опции указываются различные ключи, например, использование ключа *–v* целесообразно при просмотре нетекстового файла. В этом случае вывод “непечатных ” символов, которые могут нарушить настройки терминала, будет подавлен. Если имя файла в командной строке не указано, то ожидается вывод ввод данных со стандартного ввода. Утилита *cat* выполняет слияние и вывод файлов: по очереди читает указанные файлы и выдает их содержимое на стандартный вывод. Так, например, *cat file* распечатывает содержимое файла *file*, а *cat file1 file2 > file3* объединяет первые два файла и помещает результат в третий. Чтобы добавить файл *file1* к файлу *file2*, надо выполнить команду *cat file1 >> file2.*

Вывод содержимого файла (more)

more [-опция] файл – выводит стандартный входной поток на экран порциями по 24 строки, ожидая нажатия клавиши *Пробел* для вывода очередной порции. Досрочно завершить вывод можно, нажав клавишу Q;

По команде *more* на экран выводится содержимое текстового файла, при этом нет необходимости запускать текстовый редактор, распечатывать файл или нажимать клавишу паузы во время вывода текста на экран. Например, для вывода на экран содержимого конфигурационного файла new вводится команда: *more emacs.* Недостаток этой команды в том, что невозможно пролистать информацию в обратном направлении.

head [-n] файл – просмотреть только начало (первые n строк) файла;

tail [-опция] файл – просмотреть конец (последние n строк) файла;

less имя файла - выводит стандартный входной поток на экран порциями по 24 строки, ожидая нажатия клавиши *Пробел* для вывода очередной порции. В отличии от команды *more* поддерживает возможность прокрутки вверх и поиска; По команде *less* информация выводится в окно терминала. Имя этой команде дано в противоположность команде more, поскольку в команде less пролистывание текстового файла возможно в обоих направлениях (игра слов: more - больше, less - меньше.*).*

Поиск файлов

find имя_каталога [-ключ] - выполнить поиск файла в файловой системе, начиная с каталога ***имя_каталога***, используя различные критерии, такие как права доступа, размер, тип и т.д. Команда *find* представляет собой универсальный инструмент поиска: она позволяет искать файлы и каталоги, просматривать все каталоги в системе или только текущий каталог, проводить поиск даже на дисках NFS (Network File System), конечно, при наличии соответствующих разрешений. В подобных случаях команда обычно выполняется в фоновом режиме, поскольку просмотр дерева каталогов требует значительных затрат времени.

Символ ‘.’ в имени каталога служит для обозначения текущего каталога, символ ‘/’ - корневого каталога, а символ ‘~’ - записанного в переменной $HOME начального каталога текущего пользователя. Основные опции команды *find* представдены в таблице 1.

Таблица 1. Основные опции команды find

ключ	назначение
1	2
name	поиск файлов, имена которых соответствуют заданному шаблону
print	запись полных имен найденных файлов в стандартный поток вывода
perm	поиск файлов, для которых установлен указанный режим доступа
prune	применяется для того, чтобы команда find не выполняла рекурсивный поиск по уже найденному путевому имени; если указана опция -depth, опция -prune игнорируется
user	поиск файлов, принадлежащих указанному

	пользователю
group	поиск файлов, которые принадлежат данной группе
mtime *-m+n*	поиск файлов, содержимое которых модифицировалось менее чем (-) *m* или более чем (+) *n* дней назад; имеются также опции -atime и -ctime, которые позволяют осуществлять поиск файлов соответственно по дате последнего чтения и дате последнего изменения атрибутов файла
nogroup	поиск файлов, принадлежащих несуществующей группе, для которой, иначе говоря, отсутствует запись в файле /etc/groups
nouser	поиск файлов, принадлежащих несуществующему пользователю, для которого, другими словами, отсутствует запись в файле /etc/passwd
newer *файл* –type	поиск файлов, которые созданы позднее, чем указанный файл
	поиск файлов определенного типа, а именно:
b	специальный блочный файл;
d	каталог;
c	специальный символьный файл;
p	именованный канал;
l	символическая ссылка;
s	сокет;
f	обычный файл
size *n*	поиск файлов, размер которых составляет n единиц; возможны следующие единицы измерения:
	блок размером 512 байтов (установка по умолчанию);
b	байт;
	килобайт (1024 байта);
c	двухбайтовое слово
k	
w	
depth	при поиске файлов сначала просматривается содержимое текущего каталога и лишь затем проверяется запись, соответствующая самому каталогу
fstype	поиск файлов, которые находятся в файловой системе определенного типа; обычно соответствующие сведения хранятся в файле */etc/fstab,*
mount	поиск файлов только в текущей файловой системе; аналогом этой опции является опция –xdev
exec	выполнение команды интерпретатора shell для всех обнаруженных файлов; выполняемые команды имеют формат команда {} \; (следует обратить внимание на

	наличие пробела между символами {} и \;)
ok	аналогично exec, но перед выполнением команды отображается запрос

Например,

- name – поиск по искомому имени файла, например:

find / -name sh ,

по этой команде будет осуществляться поиск в каталоге */* файла с именем *sh;*

-**print** – обеспечивает вывод информации. Например, для вывода полного имени исполняемого файла командного интерпретатора Bourne shell, необходимо ввести команду:

find / -name sh -print 2 >/dev/null ;

Для фрагментарного поиска по имени файла (только в последней части спецификации файла), например, '*core*', следует ввести команду:

*find ~ -name '*core*' -print*

- size [размер] – поиск по заданному размеру. Например, для поиска файлов размером больше 10 Мбайт по всей файловой системе, необходимо ввести команду:

find . –size +20480 - print ;

- atime - поиск по последнему времени модификации. Например, поиск файлов с именем *file1,* обращение к которым было более *15* дней назад:

find / –name file1 -atime +15 - print ;

Для автоматического удаления всех найденных файлов с именем *core* (образ процесса, создаваемый при неудачном его завершении и используемый в целях отладки), последнее обращение к которым было более месяца (*+30*) назад, следует ввести команду:

find / –name core -atime +30 - exec rm {} \ ;

Следует отметить, что каждый раз при запуске команды, указанной после ключа *exec*, создается новый процесс. Это приводит к увеличению нагрузки на систему и излишнему потреблению ресурсов процессора и оперативной памяти. Однако при необходимости выполнить операцию, например такую как *rm,* над большим количеством файлов, эффективнее сначала построить список файлов, а затем запустить команду *rm* лишь один раз, передав ей этот список в качестве параметра.

Из приведенного выше видно, что при работе с командой *find* чаще всего используется опция *-name.* После нее в кавычках должен быть указан шаблон имени файла. Если необходимо найти все файлы с расширением txt в Вашем начальном каталоге, укажите символ '-' в качестве путевого имени. Имя начального каталога будет извлечено из переменной $HOME.

find ~ -name ".txt" -print*

Чтобы найти *все* файлы с расширением txt, находящиеся в текущем каталоге, следует воспользоваться такой командой:

find . -name ".txt" -print*

Для нахождения в текущем каталоге всех файлов, в именах которых встречается хотя бы один символ в верхнем регистре, введите следующую команду:

find . -name "[A-Z]" -print*

Команда *find . –print* аналогична команде *ls –Rfl,* но в последнем случае выводимый список будет длиннее, т.к. в процессе обхода команда ls отмечает каждый новый каталог, а команда find не обращает внимание на каталог **..** ;

grep [параметры) базовое_регулярное_выражение [файл] – поиск строки внутри файла**.** В качестве регулярного выражения может быть указана обычная строка. Если файл не указан, текст берется из стандартного входного потока. Строку, которая задана в качестве регулярного выражения и состоит из нескольких слов, следует заключать в двойные кавычки. В противном случае первое слово строки будет воспринято как образец поиска, а все остальные слова будут считаться именами файлов. В итоге отобразится сообщение о том, что указанные файлы не найдены.

Если образец поиска состоит из какой-нибудь системной переменной, например $PATH, рекомендуется тоже взять ее в двойные кавычки. Это связано с тем, что, прежде чем передавать аргументы команде grep, интерпретатор shell выполняет подстановку переменных, и команда grep получает значение переменной, которое может содержать пробелы. В этом случае будет выдано то же сообщение об ошибке, о котором говорилось в предыдущем абзаце.

Шаблон поиска должен быть заключен в одинарные кавычки, если в состав регулярного выражения входят метасимволы. Команда grep имеет следующие параметры:

-c - задает отображение только числового значения, указывающего, сколько строк соответствуют шаблону;

-i - дает указание игнорировать регистр символов;

-h - подавляет вывод имен файлов, включающих найденные строки (по умолчанию в выводе команды grep каждой строке предшествует имя файла, в котором она содержится);

-1 - задает отображение только имен файлов, содержащих найденные строки;

-n - задает нумерацию выводимых строк;

-s - подавляет вывод сообщений о несуществующих или нетекстовых файлах;

-v - задает отображение строк, не соответствующих шаблону.

which [-ключ] - поиск выполняемых файлов. Данная команда встроена в оболочку, позволяет определить точное местонахождение файла, и передает результаты своего выполнения в стандартный выходной поток. В оболочке C команда *which* позволяет определить, какие из команд являются встроенными, а какие псевдонимами.

Исследование и мониторинг системы

Для управления дисковым пространством используются команды *df, du*

и *ulimit*:

df [-ключ] – команда определяет, сколько свободного дискового пространства и индексных дескрипторов доступно в разделе смонтированного диска.

По умолчанию команда используется без параметров и выводит объем свободного пространства, например:

/ (/dev/hdb1): 260836 blocks 12034 files
/home (/dev/sda1): 260836 blocks 2104 files

В первом столбце содержится точка монтирования данной файловой системы. Затем в круглых скобках следует имя смонтированного физического устройства (в UNIX все устройства являются файлами, даже сама файловая система). Следующий столбец отображает число свободных блоков размером по 512 байт. В последнем столбце выводится количество файлов, содержащихся на данном устройстве.

При использовании ключей:

-k – вывод данных осуществляется в блоках по 1024 байт, или в килобайтах. При этом данные выводятся в формате, принятом в системе BSD:

Filesystem	**1024- blocks**	**Used**	**Available**	**Capacity**	**Mounted on**
/dev/hdb1	*1112646*	*972611*	*140035*	*88%*	*/*
/dev/sda1	*961374*	*720104*	*241270*	*75%*	*/home*

В первом столбце указано имя устройства, на котором расположена файловая система. Во втором столбце отображается размер файловой системы в блоках по 1 Кбайт. В третьем столбце выводится число используемых блоков, а в четвертом – число свободных блоков. В пятом столбце выводится процент использования диска. В последнем столбце указывается точка монтирования системы;

-P – информация отображается в формате, определенном в стандарте POSIX, который аналогичен формату, принятому в BSD;

-t - информация отображается в формате, который близок к стилю, используемому в SYSTEM V. Данные выводятся в блоках размером по 512 байт, кроме того, приводится информация, как о количестве блоков, так и о количестве индексных дескрипторов;

-i - предназначен для подсчета количества индексных дескрипторов (не поддерживается стандартом POSIX). Выводимая информация имеет следующий вид:

Filesystem	**Inodes**	**IUsed**	**IFree**	**%IUsed**	**Mounted on**
/dev/hdb1	*301056*	*93059*	*207997*	*31%*	*/*
/dev/sda1	*260096*	*17280*	*242816*	*7%*	*/home*

В качестве параметров команде *df* можно передать имя файла или список имен файлов. В этом случае отображается информация только о тех файловых системах, которые содержат указанные файлы.

du [- ключ] - команда определяет какой объем диска занимает конкретный каталог. Вызов команды без параметров позволяет получить данные о текущем каталоге. Если в качестве параметра указать имя каталога, то будет отображена информация обо всех каталогах, расположенных в иерархии ниже текущего. Если в качестве параметра указано имя файла, не являющееся каталогом, то не выводится никакой информации.

Команда *du* имеет четыре ключа:

-k – имеет то же значение, что и для команды *df* , при этом данные об использовании дискового пространства представляются в килобайтах;

-a – задает вывод данных всех перечисленных файлов. При этом полученный результат аналогичен результатам выполнения команды *ls – ls*;

-s – задает ограниченный вывод, только данные об указанном каталоге, например: *13500 /home/nata/bin,* где 13500 – размер каталога, выраженный в блоках по 512 байт;

-x – не выводятся данные о файлах, находящихся в других файловых системах. Таким образом, проверяются данные, хранящиеся в указанном каталоге локального диска;

ulimit – выводит или устанавливает значение пределов, ограничивающих использование задачей системных ресурсов (времени процессора, памяти, дискового пространства);

top – команда выдает непрерывно обновляемую таблицу всех задач, выполняющихся на компьютере, включая системные, с указанием объема используемых ресурсов. Для завершения работы команды необходимо нажать клавишу *Q*;

ps – выводит информацию о существующих процессах**.** При использовании различных опций можно получить следующую информацию:

-al - выдает в форме таблицы список пользовательских процессов, запущенных в системе;

-F – статус процесса (системный, блокировки памяти и т.д.);

-A – состояние всех процессов;

-S – состояние процесса (O – выполняется процессором, S – находится в состоянии сна, R - готов к выполнению, I - создается, Z - зомби);

- ef – распечатывает имя программы, породившей процесс, вместе со всеми параметрами;

- n name – состояние всех процессов, порожденных командами, имена которых указаны в списке *name*;

- g list – показать все процессы, запущенные пользователями групп, номера которых указаны в списке. Например, *ps –g 0* -.показать все процессы

группы 0, т.е. root. Номера групп указываются в списке через запятую или пробел.

- l – длинный формат вывода состояния процессов;

- p - состояние процессов, идентификаторы которых указаны в списке, например: *ps –p "12499, 17772"* – определить состояние процессов с идентификаторами (PID) 12499 и 17772;

w [- ключ] – команда информирует о том, что делают в системе зарегистрированные пользователи, например:

(9:12 am up 30 min, 3 users, Load average, 0.00, 0.52, 1.22)

user	*TTY*	*FROM*	*LOGIN@*	*IDLE*	*JCPU*	*PCPU*	*what*
user	*tty1*	*-*	*8.44 am*	*27:50*	*0.24s*	*0.03s*	*/bin/sh/usr*
userpts	*/0*	*-*	*8.52 am*	*29:48*	*0.00s*	*?*	*-*

Первая строка содержит текущее время, сколько времени компьютер работает без перезагрузки, число пользователей и загрузка машины. Затем следует строка, содержащая заголовки столбцов: *user* – имя пользователя, связанного с данным устройством tty; *TTY* - имя терминала (консоли); *LOGIN@* -первоначальное время регистрации; *IDLE* - количество времени, на протяжении которого пользователь ничего не вводил с клавиатуры; *ICPU* – общее время центрального процессора, использованного всеми процессами на этом терминала; *PCPU* – общее время центрального процессора для всех активных процессов на этом терминале; *what* – название и параметры текущей выполняемой команды. Далее следует список пользователей, и чем они заняты. Знак ? означает, что процесс ожидает связи с терминалом, однако в текущий момент связь отсутствует. Команда имеет три ключа:

- h – подавляет заголовки;

- l – отображает информацию в расширенном виде (используется по умолчанию);

- s – отображает информацию в краткой форме (выводятся столбцы *user, tty, iIdle, what*);

Конкретного пользователя можно проверить, введя команду

w имя _пользователя

who [-ключ]– выдается список пользователей, зарегистрированных в данный момент в системе. Например:

nata tty1 Nov 2 14:30
alex tty4 Nov 2 14:15

где - *nata* – имя пользователя,
tty1 - номера его терминала,
Nov 2 - дата,
14:30 - время подключения.

Согласно стандарту POSIX, команда должна иметь несколько ключей, влияющих на внешний вид выводимой информации:

-b – выводит время последней перезагрузки;

-d – выводит список “умерших” процессов (dead processes), которые не были повторно порождены;

-H – выводит заголовки столбцов;

-l – перечисляет номера tty, ожидающих регистрации пользователей;

-T – выводит состояние канала связи с каждым из терминалов (+ означает, что данный терминал доступен для записи, а – означает, что терминал для записи не доступен);

-t – выводит момент последнего изменения системного времени;

-s – выводит имя пользователя, tty и время регистрации в системе (используется по умолчанию);

-u – выводит время простоя для каждого терминала;

-m – выводит информацию только о текущем терминале;

-r – выводит текущее состояние системы;

-p – перечисляет все активные процессы, порожденные процессом *init;*

-g – перечисляет только пользовательские имена и количество пользователей;

Пример результата выполнения команды *who –THu :*

USER	*MESG*	*LINE*	*LOGIN-TIME*	*IDLE*
nata	+	*tty1*	*nov 10 18:44*	.
oleg	-	*tty3*	*nov 10 19:53*	*old*
alex	+	*tty4*	*nov 10 18:53*	*old*

Из примера видно, что только пользователь nata находится в активном состоянии. Пользователи oleg и alex не обращались к своим терминалам на протяжении дня. Кроме того, пользователю oleg доступ к терминалу запрещен;

last [-ключ] – позволяет определить, кто и когда зарегистрировался в системе. Для выдачи результатов она пользуется файлом /etc/utmp, в котором зафиксированы моменты входа-выхода пользователей и перезагрузки системы. При использовании команды без параметров будет выведен список в обратном порядке всех, кто работал в системе.

Для ограничения размера списка в качестве параметра следует указать некоторое число, например,

last -25

выводит список последних 25 пользователей. Введя команду *last reboot,* можно просмотреть список последних перезагрузок;

finger – команда позволяет определить, находится ли в системе некоторый пользователь. Введя команду

finger – имя_пользователя

можно получить разнообразную информацию, включающую и время последней регистрации данного пользователя в системе;

at [-ключ] время_запуска - считывает команды стандартного потока ввода и группирует их в задания *at,* которое будет выполнено в указанное пользователем время. . Например:

at now + 2minutes

Для выполнения задания будет запущен командный интерпретатор, в среде которого и будут исполнены команды.

uptime – позволяет оценить стабильность и загрузку системы. Данная команда выводит только первую строку информации команды *w*, например,

9:12 pm up 10 days, 10:51, 4 users, load average: 0.01, 0.03, 0.22)

2.4.8. Менеджер файлов Midnight Commander (mc)

Программа Midnight Commander - полифункциональный менеджер файлов, работающий в текстовом режиме (т.е. в текстовой консоли или терминале). Интерфейс программы похож на двух панельные менеджеры файлов Norton Commander для MS-DOS, FAR и Windows Commander для ОС Windows, а по набору функций не уступает лучшим из них. Файловые операции mc выполняются аналогично. Структура файловой системы приведена на рис.2

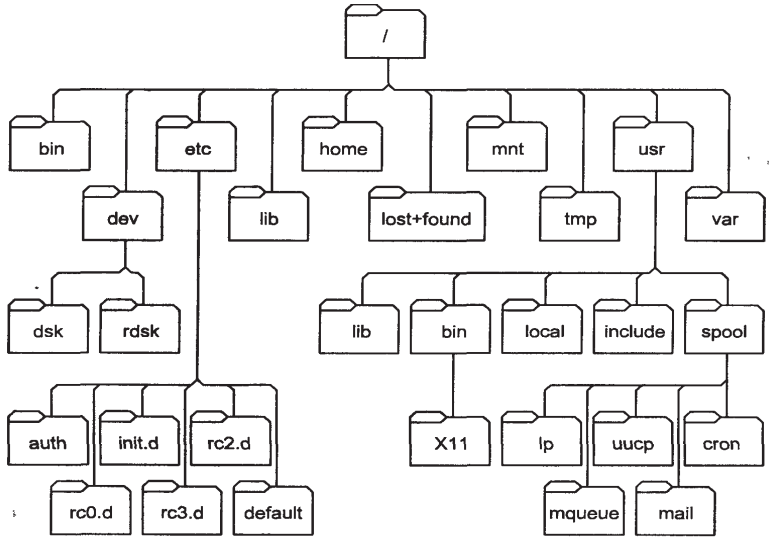

Рисунок 2. –Структура файловой системы

2.4.9. Простейшие текстовые редакторы

Для работы в текстовой консоли ОС можно воспользоваться несколькими простейшими текстовыми редакторами, которые позволяют

изменить конфигурационный файл системы или набрать текст сценария. В текстовом режиме, как и в оболочке KDE, можно использовать профессиональную систему подготовки текста *emacs* (включающую в качестве макроязыка язык программирования высокого уровня), однако ее рассмотрение выходит за пределы данного пособия.

В простых случаях можно воспользоваться встроенным редактором программы *Midnight Commander (mc).* Для того чтобы отредактировать текстовый файл во встроенном редакторе *mc*, выберите нужный файл в активной панели и нажмите клавишу *F4*. В открывшемся окне редактора можно вводить или редактировать текст. При необходимости следует использовать кнопки операций с блоками текста или поиска по образцу или, нажав клавишу *F9,* открыть меню, позволяющее устанавливать пользовательские настройки редактора, или осуществлять такие операции, как форматирование текста и обработка при помощи макросов.

Существующий несколько десятков лет тестовый редактор *vi* имеет специфическую систему команд и сохраняется в современных системах UNIX (Linux) во многом лишь по традиции. Однако, некоторые старые командные файла (скрипты), могут по умолчанию вызывать данный редактор для редактирования файлов пользователя. В этом случае понадобиться выйти из текстового редактора *vi* без сохранения изменений: поместить курсор с помощью клавиши *Backspace* в ту часть окна, где расположен текст; далее набрать символ *:* (нажав клавиши *Shift* - *:*), курсор вместе с набранным символом переместиться в нижнюю строку экрана (поле команд); ввести в этом поле последовательность символов *q!* и нажать клавишу *Enter.*

Для создания и редактирования файлв в редакторе vi необходимо выполнить следующие действия:

1) Перейти в командный режим, нажав клавишу :

2) Создать новый файл, например file, с помощью команды:*edit file* . Так как такого файла еще нет, то в нижней строке редактора появится сообщение *"file" [New File].* Вначале файл file пуст и на экране нет букв или цифр.

3) Для начала ввода текста в файл существуют две команды:

- i - текст добавляется перед текущим символом (курсор указывает на этот символ);
- a - текст добавляется после текущего символа.

Для перемещения по тексту в режиме ввода текста нельзя использовать мышь и клавиши управления курсором, так как вместо сдвига курсора будут вводиться служебные символы. Для передвижения курсора служат специальные комбинации клавиш, о который можно узнать с помощью команды *:help*.

Для более удобного перемещения указателя ввода следует воспользоваться клавишей *Esc* - перейти в режим просмотра текста. В этом режиме работают клавиши управления курсором, поэтому можно выбрать

нужную позицию в тексте и нажатием клавиши с символами *i* или *a* снова перейти в режим ввода текста.

4) В режиме просмотра текста допускается удалять текущий символ с помощью команды *x*. Для удаления всей строки используется команда *dd*.

5) В режиме ввода текста работает клавиша *Backspase*, но только в текущей строке, причем стираемые символы продолжают оставаться на экране!

6) Для сохранения файла следует нажать на клавишу *Esc*, а потом ввести символ *:* для перехода в командный режим.

7) После появления внизу экрана приглашения командного режима, необходимо ввести символ *w*, т.е. дать команду для записи файла. Если процесс записи файла закончится успехом, появится информация о записанном файле, а также статистика, например:

"file " [New File] 5 lines. 65 characters written

Существует множество опций, которые влияют на работу vi. Все доступные опции можно посмотреть с помощью команды *set all*. Также команду *set* можно использовать для установки различных опций. Например, для отображения порядковых номеров строк файла, нужно использовать команду: *:set number*

Для отключения отображения номеров строк необходимо ввести команду:

:set nonumber

Некоторые опции могут иметь параметры, например, *:set tabstop=4* заставит символ табуляции отображаться как четыре пробела, вместо обычных восьми.

Для сохранения опций и при следующем сеансе работы, необходимо поместить их в файл *.exrc*, или установить переменную окружения EXINIT соответствующим образом. Например, если в качестве shell используется Borne shell, можно разместить в вашем .profile следующую строчку:

EXINIT='set nomagic nu tabstop=4'; export EXINIT

Если используется C shell, необходимо разместить в вашем .login файле:

setenv EXINIT='set nomagic nu tabstop=4'

2.4.10 Монтирование файловой системы

Для того чтобы сделать доступными файловые системы, размещенные на физических устройствах, требуется совместить корневые каталоги этих устройств некоторыми подкаталогами файловой системы. Только после этого ядро сможет выполнять файловые операции такие как, создание, открытие, чтение/запись в файл. Операция встраивания получила название *подключение* или *монтирование файловой системы.* Жесткие диски обычно монтируются при запуске системы (например, основной раздел, куда Вы установили ОС Linux, монтируется корневой каталог /), однако их можно монтировать и отдельно. Другие устройства, например, дисководы для сменных дисков (устройство /dev/fdO, /dev/fdyl), дисковод CD-ROM (/dev/cdrom), монтируются по мере необходимости. По окончании работы с устройствами они

размонтируются, и размещенные них файлы становятся недоступными. Для жестких дисков это действие обычно выполняется автоматически при отключении системы. Для CD-ROM соответствующие устройства необходимо размонтировать перед тем, как удалять из компьютера носители. Нельзя удалить компакт-диск из неразмонтированного привода (устройство не реагирует на кнопку извлечения); вынуть сменны носитель из не размонтированного дисковода физически возможно, однако это приводит к полному уничтожению информации на ней.

Монтирование некоторых устройств можно осуществлять средствами GNOME, KDE и т.п. (рис.3) , но более гибкое решение предоставляет текстовая консоль.

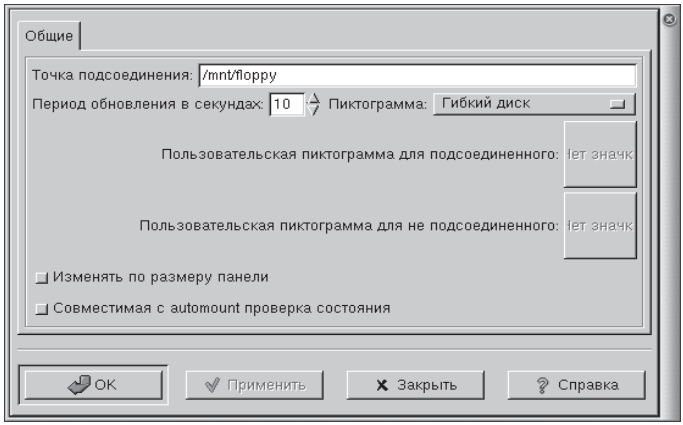

Рисунок 3- Вид панели монтирования в графической оболочке

Монтирование файловой системы осуществляется системным вызовом *mount*:

mount тип монтируемой системы точка монтирования флаги и дополнительные данные

где *точка монтирования* – это имя каталога, к которому подключается файловая система;

тип_монтируемой_файловой_системы — ключевое слово, обозначающее стандарт, в котором записываются на устройстве файлы каталоги. Чаще всего могут встретиться следующие типы файловых систем:

ext2 — файловая система, используемая в разделах жесткого диска Linux;

iso9660 — файловая система, используемая на большинстве CD-ROM;

Таким образом, чтобы смонтировать привод CD-ROM в каталог /mnt, необходимо ввести команду

mount -t iso9660 /dev/cdrom /mnt

2.5. Вопросы к защите лабораторной работы

1) Перечислите особенности работы в текстовой и графической консолях.
2) Перечислите пользователей системы.

3) Как добавить пользователя, группу пользователей в систему?
4) Какая команда используется для изменения владельца текущего сеанса?
5) Какую информацию содержат файлы /etc/shell, /etc/passwd?
6) Где и в каком поле записи указывается командный интерпретатор пользователя?
7) Как можно изменить режим доступа к файлам?
8) Поясните поля записи из файла /etc/passwd.
9) Перечислите файлы, относящиеся к служебным учетным записям.
10) Укажите формат команды для получения информации о процессах, связанных с терминалом.
11) Укажите формат команд, используемых для исследования системы.
11) Перечислите команды для идентификации файлов.
12) Что понимается под монтированием файловой системы?
13) Как осуществляется монтирование устройств?
14) Приведите формат команды монтирования/размонтирования устройств.
15) Перечислите этапы монтирования системы при загрузке ОС.

2.6. Список рекомендуемой литературы

1) Робачевский А. Операционная система UNIX.-СПб.:BHV-Санкт-Петербург, 2020- 528 с.

2) Романчева Н.И. ОС Linux – принципы, технологии, инструменты/ Учебное пособие.- М.: МГТУ ГА, 2011.

3. ЛАБОРАТОРНАЯ РАБОТА №2
ОС LINUX: ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ КОМАНДНОГО ИНТЕРПРЕТАТОРА

3.1. Цель работы

Целью данной работы является получение практических навыков получение практических навыков и умений программирование на языке командного интерпретатора, получение навыков отладки сценариев.

3.2. Перечень компетенций, формируемых в ходе выполнения лабораторной работы

Общепрофессиональная компетенция ОПК-2:

Способен применять информационно-коммуникационные технологии, программные средства системного и прикладного назначений, в том числе, отечественного производства, для решений задач профессиональной деятельности.

- **умеет:** создавать и отлаживать скрипты;

-владеет навыками создания простых файлов-сценариев на языке Born-shell.

3.3. Задание на выполнение работы

1) Написать и отладить программу на языке интерпретатора Borne:
   - используя команду echo;
   - используя команду управления форматированием вывода (printf);
2) Написать и отладить программу, обрабатывающую ввод из потока STDIN;
3) Написать и отладить программу, считывающую данные из потока и использующую несколько переменных в аргументе команды read. При запуске сценария задать число переменных в списке меньше (больше) числа аргументов, считанных из потока.
4) Написать сценарий для выполнения проверки значений введенных переменных.
5) Написать и отладить программу вычисления длины окружности и площади круга.
6) Создать файл функций *functions.main*, включающий одну функцию, которая будет загружена интерпретатором команд, протестирована, изменена, а затем повторно загружена. Файл должен содержать операторы вывода количества параметров, задаваемых при запуске файла, название файла –сценария.

3.4. Основные приемы работы

3.4.1. Программирование на языке командного интерпретатора

Интерпретатор (shell) представляет собой интерфейс командной строки. По своей функциональности он похож на интерпретатор COMMAND.COM системы MS DOS. Одной из задач интерпретатора является обеспечение безопасного и структурированного доступа к ядру UNIX-подобных

операционных систем, т.е. фактически это программный уровень, который предоставляет среду для ввода команд, обеспечивая тем самым взаимодействие между пользователем и ядром операционной системы. Кроме того, встроенный в интерпретатор мощный язык программирования используется для решения различных задач: от автоматизации повторяющихся команд до написания сложных интерактивных программ обработки данных, получения информации из небольших баз данных.

В среде UNIX существует много командных интерпретаторов, среди которых можно выделить такие как sh, tcsh, ksh, csh, bash, в Linux по умолчанию используется bash.

3.4.2. Встроенные команды интерпретатора shell

: Нуль, всегда возвращает истинное значение.

. Считывание файлов из текущего интерпретатора shel.l

break Применяется в конструкциях for, while, until, case.

cd Изменяет текущий каталог.

continue Продолжает цикл, начиная следующую итерацию.

echo Записывает вывод в стандартный поток вывода.

eval Считывает аргумент и выполняет результирующую команду.

exit Выход из интерпретатора shell.

export Экспортирует переменные, вследствие чего они доступны для текущего интерпретатора shell.

pwd Отображает текущий каталог.

read Просматривает строку текста из стандартного потока.

readonly Превращает данную переменную в переменную "только для чтения".

return Выход из функции с отображением кода возврата.

set Управляет отображением различных параметров для стандартного потока вводных данных.

shift Смешает влево командную строку аргументов.

test Оценивает условное выражение

times Отображает имя пользователя и системные промежутки времени для процессов, которые выполняются с помощью интерпретатора shell

trap При получении сигнала выполняет определенную команду.

type Интерпретирует, каким образом интерпретатор shell применяет имя в качестве команды.

ulimit Отображает или устанавливает ресурсы интерпретатора shell.

umask Отображает или устанавливает режимы создания файлов, заданные по умолчанию.

unset Удаляет из памяти интерпретатора shell переменную или функцию.

wait Ожидает окончания дочернего процесса и сообщает о его завершении.

Сам интерпретатор shell автоматически присваивает значения следующим переменным (параметрам):

? - значение, возвращенное последней командой;
$ - номер процесса;
! - номер фонового процесса;
 # - число позиционных параметров, передаваемых в shell;
***** - перечень параметров, как одна строка;
@ - перечень параметров, как совокупность слов;
- флаги, передаваемые в shell.

При обращении к этим переменным, т.е. при использовании в командном файле, следует впереди ставить "$".

Важную роль при создании уникальных файлов играет специальная переменная "$$", значение которой соответствует номеру процесса, выполняющего данный расчет. Каждый новый расчет, выполняемый компьютером, инициирует один или несколько процессов, автоматически получающих номера по порядку. Поэтому, используя номер процесса в качестве имени файла, можно быть уверенным, что каждый новый файл будет иметь новое имя (не запишется на место уже существующего). Достоинство является и главным недостатком такого способа именования файлов. Неизвестно, какие имена будут присвоены файлам. И, если в рамках данного процесса можно найти файл "не глядя", т.е., обратившись к нему, используя $$, то потом такие файлы можно легко потерять. Это создает дополнительные проблемы при отладке программ.

3.4.3. Приемы экранирование

Рассмотрим приемы экранирования, используемые в shell. В качестве средств экранирования используются двойные кавычки (" "), одинарные кавычки (' ') и бэк-слэш (\).

Экранирующий символ (\), используемый совместно с командами *echo* и *sed,* сообщает интерпретатору, что следующий за ним символ должен восприниматься как специальный символ:

\n - перевод строки (новая строка);
\r - перевод каретки;
\t - табуляция;
\v - вертикальная табуляция;
\b - забой (backspace);
\a - "звонок" (сигнал);
\0xx - ASCII-символ с кодом *0xx* в восьмеричном виде.

В командном файле бэк-слэш экранирует конец строки, что позволяет объединять строки в одну, т.е. запись:

*cat file | grep -h *
result | sort | cat -b > filerez

аналогична записи:

cat file | grep -h result | sort | cat -b > filerez

3.4.5. Примеры программ

Ниже приводятся примеры нескольких способов написания простой программы вывода фразы "Hello, Linux!" на языке интерпретатора Bourne.

Пример 1.

```
#!/bin/sh
# Первая программа "Hello, Linux!",
# реализованная для интерпретатора Bourne
echo
echo   " Hello, Linux! "
echo
exit   0
```

Пример 2.

```
#!/bin/sh
# Программа, "с командой printf",
# реализованная для интерпретатора Bourne
printf    "\n" с командой printf\n\n!"
exit   0
```

Пример 3.

```
#!/bin/sh
# Программа, "воспринимающая ввод с клавиатуры",
# реализованная для интерпретатора Bourne
echo
echo   -n   "Введите name:    "
read   name
echo
echo   "Hello,    ${name}!"
echo
exit   0
```

В примерах 1-3 первые строки содержат последовательность символов #!, сообщающую ОС, что за ней следует имя командного интерпретатора, в котором сценарий должен выполняться. В данном случае это интерпретатор Bourne, /bin/sh. Строки, начинающиеся с символа #, являются комментариями, и интерпретатор их игнорирует. Команда *echo* выводит свои аргументы в стандартный вывод (обычно это экран). Команда *exit* завершает работу программы и возвращает код выхода родительской программе (обычно это командный интерпретатор). Код завершения, равный 0, указывает, что программа нормально завершила работу. Код, отличный от 0, сообщает об ошибке. Если статус завершения не указан явно, то программа возвращает код завершения последней выполненной команды.

Простого вывода сообщений на экран недостаточно для решения задач, поэтому язык сценариев использует переменные. Все переменные в программах командного интерпретатора хранятся как строки. Кроме присвоения значений переменным и их использования в сценарии, командный интерпретатор предоставляется возможность обрабатывать ввод из потока STDIN. Для чтения пользовательского ввода используется команда *read.* Команда *read* в качестве аргументов может иметь несколько переменных (пример 4). В этом случае всякий пустой символ трактуется как символ разделитель между значениями, присваиваемыми разным переменным. Если число аргументов, прочитанных из потока ввода, меньше, чем число переменных в списке, оставшимся переменным не присваивается никаких значений. Если аргументов больше, чем переменных, то все оставшиеся значения присваиваются последней переменной.

Пример 4.

```
#!/bin/sh
echo
echo -n "Введите три числа, разделенные пробелами или символом табуляции: "
read var 1 var2 var3
echo
echo "The of var1 is: ${var1}"
echo "The of var2 is: ${var2}"
echo "The of var3 is: ${var3}"
echo
exit 0
```

Для операций с числами с плавающей запятой, а также обработки сложных выражений, где порядок операций изменен, применяется команда *bc* (пример 5).

Пример 5.

```
#!/bin/sh
# Данная программа вычисляет длину окружности
# и площади круга
pi="3.14159265"        # Переменной присваивается число p
# пользователю выводится сообщение и запрашивается радиус
# окружности
echo
echo -n "Введите радиус: "
read radius
# Вычисляемые значения сохраняются в переменных
cir=$[ echo $radius*2*$pi | bc ]
area= $[ echo $radius^2*$pi | bc ]
```

```
#Программа выводит результаты и завершает работу
printf   "\n\nThe circumference is:\t$cir\n"
printf   "The area is:\t\t$area\n\n"
exit  0
```

Сценарий не относится к компилируемым программам, поскольку он интерпретируется построчно. После ввода текста программы и сохранения, необходимо сделать файл выполняемым. Для этого используется команда:

chmod u+x имя файла-сценария

Эта команда устанавливает права владельца на запуск файла *text.* Для запуска файла необходимо ввести в командной строке: *. /text*

Пример 6.

```
#!/bin/sh
echo -n "What is your first name :"
read F_NAME
echo -n "What is your surname :"
read S_NAME
printf   "\n\n What is your first name :\t$ F_NAME \n"
exit 0
```

Данный сценарий предназначен для выполнения проверки значений переменных (переменным присваиваются исключительно одни символы).

Пример 7.

```
#!/bin/sh
#  functions.main
#  findit: интерфейс для базовой команды find
findit ()
# findit
if [ $# -It 1 ]; then
echo "usage :findit file"
return 1
fi
find / -name $1 –print
exit 0
```

В данном примере создается файл функций *functions.main*, включающий одну функцию. Эта функция будет загружена интерпретатором команд, протестирована, изменена, а затем повторно загружена. Данный код лежит в основе интерфейса для базовой команды find. Если команде не передаются аргументы, то возвращается значение 1 (что свидетельствует, о возникновении ошибки). Обратите внимание, что ошибочная конструкция фактически является отображенным именем функции (если же была использована команда $0, интерпретатор команд просто возвращает сообщение sh). Причина отображения

подобного сообщения заключается в том, что файл не является файлом сценария.

Порядок передачи параметров функции аналогичен передаче параметров обычному сценарию. При этом используются специальные переменные $1, $2, ... $9. При получении функцией переданных ей аргументов происходит замена аргументов, изначально переданных сценарию интерпретатора shell. В связи с этим желательно было бы повторно присвоить значения переменным, получаемым функцией. В любом случае это стоит сделать, так как при наличии ошибок в функциях их можно будет легко обнаружить, воспользовавшись именами лекальных переменных. Для вызывающих аргументов (переменных), находящихся внутри функции, имя каждой переменной начинается с символа подчеркивания, например: _FILENAME или _filename.

После естественного завершения выполнения функции, либо в том случае, когда она завершается в результате выполнения какого-либо условия, можно выбрать один из двух возможных вариантов:

1) завершение функции и последующая передача управления той части сценария, которая вызвала данную функцию.

2) использование ключевого слова return, в результате чего будет осуществлена передача управления конструкции, которая расположена за оператором вызова функции. При этом может также указываться необязательный числовой параметр. Этот параметр принимает значение 0 в случае отсутствия ошибок и значение 1 - при наличии ошибок. Действие этого параметра аналогично действию кода завершения последней команды.

При использовании ключевого слова return применяется следующий формат:

return - возвращает результат из функции, использует код завершения последней команды для проверки состояния;

return 0 - применяется при отсутствии ошибок;

return 1 - применяется при наличии ошибок.

Внимание! В приведенных примерах имеются синтаксические ошибки, которые необходимо устранить.

3.5. Вопросы к защите лабораторной работы

12) Что понимается под термином “пользовательская среда UNIX”?
13) Поясните общую структуру скрипта.
14) В каком виде хранятся переменные в программах командного интерпретатора?
15) Приведите примеры сложных синтаксических конструкций получения значений переменной.
16) Перечислите внутренние переменные shell, используемые в скриптах.
17) В каких случаях используется команда bc ?
18) Поясните процедуру создания файла функций.
19) Поясните назначение формальных параметров.

20) Как осуществляется подстановка позиционных параметров?

3.6. Список рекомендуемой литературы

1) Робачевский А. Операционная система UNIX.-СПб.:BHV-Санкт-Петербург, 2020- 528 с.

2) Романчева Н.И. ОС Linux – принципы, технологии, инструменты/ Учебное пособие.- М.: МГТУ ГА, 2011.

4. ЛАБОРАТОРНАЯ РАБОТА №3
ОС LINUX: РАСШИРЕННЫЕ ВОЗМОЖНОСТИ КОМАНДНЫХ ИНТЕРПРЕТАТОРОВ

4.1. Цель работы

Целью данной работы является получение практических навыков и умений программирования на языке командного интерпретатора, а также навыки отладки программ на языке С для операционной системы с открытым кодом на примере ОС UNIX/LINUX.

4.2. Перечень компетенций, формируемых в ходе выполнения лабораторной работы

Универсальная компетенция УК-2:

Способен осуществлять критический анализ проблемных ситуаций на основе системного подхода, вырабатывать стратегию действий:

- умеет устанавливать и изменять базовые классы и типы прав доступа к файлу; использовать условные операторы и операторы цикла в скриптах; использовать shell-функции;

Общепрофессиональная компетенция ОПК-2:

Способен применять информационно-коммуникационные технологии, программные средства системного и прикладного назначений, в том числе, отечественного производства, для решений задач профессиональной деятельности:

- владеет технологией перехвата прерываний.

4.3. Задание на выполнение работы

1) Просмотреть значение переменной окружения и перейти в домашнюю директорию.

2) Установить переменную окружения.

3) Написать и отладить скрипт, осуществляющий поиск заданного слова в файле паролей и содержащий вывод комментарий к командам.

4) Написать и отладить скрипт, иллюстрирующий различные способы защиты файлов. Для контроля (чтения) использовать команду cat:

- установить для файла только право на запись;
- для каталога установить только право на чтение;
- вывести список и прочитать файл (должно быть - файлы видно, нельзя прочитать содержимое);
- установить для каталога бит использования (выполнения, execute bit);
- прочитать содержимое файла и удалить (должно быть - файлы читаются, но не удаляются);
- установить корректно права на доступ к каталогу и файл удалить.

5) Написать и отладить скрипт для проверки количества аргументов командной строки, переданной программе. Если их число больше или равно 1, программа должная выполнять операции внутри блока. Если они отсутствуют, программа завершает работу, не выполняя никаких действий.

6) Используя условные выражения, написать и отладить скрипт, проверяющий наличие в домашнем каталоге инициализационного скрипта .profile, а в случае его отсутствия выполнить копирование шаблона.

7) Написать скрипт очистки неиспользуемых файлов (за определенный промежуток времени) во временных каталогах (/tmp, /usr/tmp), в каталоге /home/user.

8) Используя операторы цикла while, until, написать скрипт, выводящий на экран целые числа в заданном диапазоне.

9) Написать и отладить программу, демонстрирующую перехват прерываний при выходе.

10) Разработать функцию, удаляющую временные файлы при завершении работы сценария.

4.4. Основные приемы работы

4.4.1. Переменные окружения

Окружение/среда (environment) - это набор пар ПЕРЕМЕННАЯ=ЗНАЧЕНИЕ, которые могут использоваться запускаемыми процессами. Для того чтобы посмотреть свое окружение введите команду без аргументов:

env

В зависимости от конфигурации системы, вывод списка переменных окружения может занять несколько экранов, поэтому лучше использовать команду с постраничным просмотром:

env|more

4.4.2. Организация циклов и ветвлений

а) Условный оператор "if"

В общем случае оператор "if" имеет структуру:

```
if условие
        then список
        [elif условие
                then список]
        else список]
fi
```

Служебное слово "elif" - сокращенный вариант от "else if", может быть использован наряду с полной записью, т.е. допускается вложение произвольного числа операторов "if" (как и других операторов). Если выполнено условие, то выполняется "список", иначе он пропускается. Структура обязательно завершается служебным словом "fi". Число "fi" всегда должно соответствовать числу "if".

б) Команда test

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

test условие

или

[условие]

Интерпретатор shell будет распознавать эту команду по открывающей скобке "[". Между скобками и содержащимся в них условием обязательно должны быть пробелы. Пробелы должны быть и между значениями и символом сравнения или операции. В shell используются условия различных "типов":

1) 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".

2) сложные условия, реализуемые с помощью типовых логических операций:

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

3) 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.

4) 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).

в) Оператор цикла с перечислением – for

Оператор цикла "for" имеет структуру:

for имя in список-значений
do
список команд
done

где *for* - служебное слово, определяющее тип цикла, *do* и *done* - служебные слова, выделяющие тело цикла. Фрагмент *in список-значений* может отсутствовать.

Рассмотрим пример:

```
for i in f1 f2 f3
        do
        procsort $i
done
```

В этом примере имя *i* играет роль параметра цикла. Это имя можно рассматривать как shell-переменную, которой последовательно присваиваются перечисленные значения (i=f1, i=f2, i=f3), и выполняется в цикле команда *procsort*.

Часто используется форма *for i in **, означающая для всех файлов текущего каталога.

г) Оператор цикла с истинным условием - while

Структура оператора while предпочтительнее тогда, когда неизвестен заранее точный список значений параметров или этот список должен быть получен в результате вычислений в цикле. Оператор цикла while имеет структуру:

```
while
        условие
        do
        список команд
        done            ,
```

где *while* - служебное слово, определяющее тип цикла с истинным условием. Список команд в теле цикла (между *do* и *done*) повторяется до тех пор, пока сохраняется истинность условия (т.е. код завершения последней команды в теле цикла равен 0) или цикл не будет прерван изнутри специальными командами (break, continue или exit). При первом входе в цикл условие должно выполняться.

д) Оператор цикла с ложным условием until

Отличие оператора цикла *until* от оператора *while* состоит в том, что условие цикла проверяется на ложность (на ненулевой код завершения последней команды тела цикла) после каждого (в том числе и первого) выполнения команд тела цикла. Программистов, знакомых с операторами *until* в других языках может вначале сбивать такая семантика этого оператора.

Оператор цикла *until* имеет структуру:

```
until условие
        do
        список команд
        done
```

где *until* - служебное слово, определяющее тип цикла с ложным условием. Список команд в теле цикла (между *do* и *done*) повторяется до тех пор, пока

сохраняется ложность условия или цикл не будет прерван изнутри специальными командами (break, continue или exit). При первом входе в цикл условие не должно выполняться.

Кроме рассмотренных операторов ветвления и циклов существует ряд других операторов, которые можно найти в приведенной в списке литературе.

4.4.3. Функции интерпретатора shell

Интерпретатор команд shell позволяет группировать наборы команд или конструкций, создавая повторно используемые блоки. Подобные блоки называются shell-функциями.

Функция состоит из двух частей:

Метка функции Тело функции

В качестве метки выступает имя функции, тело функции образует набор команд, составляющих саму функцию. Имя функции должно быть уникальным это связано с тем, что в случае наличия двух различных функций с одинаковыми именами сценарий просто не сможет вызвать нужную функцию.

Формат, применяемый для определения функций, имеет следующий вид:

имя функции ( )
{
команда1
...
}

Можно также использовать ключевое слово function перед именем функции:

function имя_функции ( )
{
}

Функцию можно представлять себе как некоторый вид сценария, находящегося внутри другого сценария, но в этом случае следует учитывать одну особенность. При вызове функции она остается в текущем интерпретаторе shell, а ее копия хранится в памяти. С другой стороны, если вызывается или выполняется сценарий из другого сценария, создается отдельный интерпретатор shell. В таком случае становятся недействительными все существующие переменные, определенные в предыдущем сценарии.

Функции могут быть размещены в том же самом файле, что и сценарии, либо в отдельном файле, содержащем функции. При этом функции не всегда включают множество конструкций или команд - может просто содержаться единственная конструкция echo, которая выполняется при вызове функции.

Применение функций позволяет сэкономить массу времени, затрачиваемого на разработку сценариев. Благодаря созданию универсальных и многократно используемых функций отпадает необходимость в технической поддержке разработанных сценариев, использующих эти функции.

4.4.4. Объявление функций в сценарии

Перед использованием функций их необходимо объявить. Суть объявления заключается в том, что все функции должны быть размещены в

начале кода сценария. Невозможно сослаться на функцию до тех пор, пока она не попадет в "поле зрения" интерпретатора команд. Для вызова функции требуется просто ввести ее имя, например:

```
hello  ()
(
echo "Hello"
}
```

В этом примере функция называлась "hello", тело функции включало конструкцию echo, которая, в свою очередь, отображала слово.

4.4.5. Передача параметров функции

Порядок передачи параметров функции аналогичен передаче параметров обычному сценарию. При этом используются специальные переменные $1, $2, ... $9. При получении функцией переданных ей аргументов происходит замена аргументов, изначально переданных сценарию интерпретатора shell. В связи с этим желательно было бы повторно присвоить значения переменным, получаемым функцией. В любом случае это стоит сделать, так как при наличии ошибок в функциях их можно будет легко обнаружить, воспользовавшись именами лекальных переменных. Для вызывающих аргументов (переменных), находящихся внутри функции, имя каждой переменной начинается с символа подчеркивания, например: _FILENAME или _filename.

4.4.6. Перехват прерываний при выходе

Как известно, выполнение программы можно прервать, послав какой-либо из сигналов kill, а также посредством различных комбинаций клавиш (например, Ctrl-C). Проблема заключается в том, что если программа создает временные файлы, а пользователь прерывает ее работу по Ctrl-C, созданные файлы не будут удалены. За короткий срок таких файлов может накопиться достаточно много. Командный интерпретатор позволяет перехватывать подобные прерывания. Ниже приведен короткий пример, иллюстрирующий перехват прерываний:

```
#!/bin/sh
# Программа демонстрирует перехват прерываний
trap 'echo "Interrupt received. Quitting." 1>&2' 1 2 3 15
echo  -n "Enter a number: "
readln num
exit 0
```

Программа устанавливает перехват прерываний 1, 2, 3 и 15. Предпринимаемые действия задаются в одинарных кавычках. Если при запуске программы, после того как она выдаст приглашение "Enter a number", нажать Ctrl-C, она получит сигнал 2 (INT). Поскольку программа перехватывает это прерывание, она выдаст сообщение "Interrupt received" и завершит работу.

В приведенном примере следует обратить внимание на использование команды echo. Фактически, это перенаправление вывода, обеспечиваемое самим интерпретатором. Команда 1>&2 в операторе echo перенаправляет

вывод в поток STDERR. Поэтому специальное сообщение нельзя случайно перенаправить другой команде с помощью конвейера или в файл вместе с остальным выводом программы. Рекомендуется все сообщения об ошибках перенаправлять в поток STDERR командой 1>&2.

Перехват прерываний обычно используется для операций удаления временных файлов. Если требуется выполнить несколько действий, желательно воспользоваться функцией, перехватывающей прерывание. Если необходимо исключить возможность выхода из программы по Ctrl-C, можно установить перехват прерывания, не исполняющий никаких действий. Например:

```
trap  ''  2
```

Данный оператор приводит к тому, что сигнал 2 полностью игнорируется. Программа может перехватывать несколько прерываний и выполнять различные действия в зависимости от того, как осуществляется выход из нее. Перехват сигнала 0 установлен для всех вариантов выхода из программы. Перехват остальных сигналов осуществляется лишь тогда, когда они посылаются программе. Список наиболее часто перехватываемых сигналов приведен ниже:

0 - выход; 1 – HUP – обрыв сеанса (или отсоединение);
2 – INT – прерывание (Ctrl-C); 3 – Quit - выход (Ctrl -\)$
15 – TERM –обычная команда kill.

Сигнал 15 (посылаемый по умолчанию командой kill) и другие сигналы команды kill можно перехватывать, однако это не относится к сигналу 9 (SIGKILL). Он используется в качестве последней возможности прервать работу программы, когда остальные методы не помогают. Поэтому его нельзя ни перехватить, ни игнорировать.

4.5. Примеры выполнения

1) Просмотр значения переменной окружения

Переменные окружения могут формироваться как из заглавных, так и из строчных символов, однако исторически повелось именовать их в верхнем регистре.

Чтобы вывести на экран значение какой-нибудь переменной окружения, достаточно набрать *echo $ИМЯ_ПЕРЕМЕННОЙ*, например, просмотр домашней директории пользователя, хранящийся в переменной окружения ***$HOME:***

```
echo $HOME
```

2) Переход в домашнюю директорию

Для перехода в домашнюю директорию следует использовать команду:

```
cd $HOME
```

3) **Установка переменной окружения**

Для установки значений переменной окружения введите команду:

```
VAR=value
```

где VAR — название переменной;
value — значение переменной.
После вывода результата на экран:
echo $VAR
получаем заданное значение valua

3) Использование команды grep, например:

if grep user /etc/passwd>/dev/null 2>&1

then

echo пользователь user найден в файле паролей

fi

4) **Скрипт, иллюстрирующий различные способы защиты файлов.** Выполните и внесите в отчет результат выполнения данной последовательности команд:
…….

mkdir testdir
echo some data > testdir/testfile
ls -l testdir/testfile
cat testdir/testfile
chmod 0200 testdir/testfile
cat testdir/testfile
chmod 0644 testdir/testfile
chmod 0444 testdir
ls testdir
cat testdir/testfile
chmod 0544 testdir/testfile
rm testdir/testfile
chmod 755 testdir
rm testdir/testfile
rm testfile

5) Скрипт для проверки количества аргументов командной строки, переданной программе.

а) использование оператора if

```
1.  #!/bin/sh
2.  # Пример использования оператора if
3.  if [ $# -ge 1 ]
4.  then
5.      echo "You supplied $# command line arguments."
6.  fi
7.  echo
```

```
8.    echo   "Program  exiting…"
9.    echo
10.   exit  0
```

Пример запуска программы:

```
./test   file1  file2   file3
```

где *test* – имя файла (скрипта), *file1 file2 file3* – аргументы.

Оператор if в строке 3 проверяет число аргументов командной строки, используя переменную $#, в которой оно содержится. Если оно больше или равно 1, выполняются операторы, заключенные между ключевыми словами then и fi (fi - это if наоборот, данный оператор отмечает конец блока). Если нет, управление передается оператору, следующему за fi. В данном случае он информирует пользователя о выходе из программы.

Часть then оператора if является обязательной, а часть else — нет. В рассмотренном примере операторы в блоке then выполняются тогда, когда условие истинно.

б) использование команды test

```
#!/bin/sh
# Пример использования команды test
if   [ $myvar -gt 5 ]
then
        : # Не предпринимать никаких действий, выйти из блока if else
        # Операторы,  которые выполняются,  если условие ложно.
fi
exit  0
```

6) **Скрипт, проверяющий наличие в домашнем каталоге инициализационного скрипта .profile**

Создайте и запустите скрипт, фрагмент которого приведен ниже:

```
#!/bin/sh

 if [ ! -f  $HOME/.profile ]
 then
       echo "файла нет - скопируем шаблон"
       cp   /home/user123/.profile   $HOME/.profile
fi
exit  0
```

7) **Скрипт очистки неиспользуемых файлов.**

Создайте и запустите скрипт, фрагмент которого приведен ниже:

```
#!/bin/sh
   for   dir   in    /tmp   /usr/tmp  /home/user
   do
      find $dir !  -type  d  -atime  +11  -exec rm {} \;
```

```
done
exit 0
```

8) **Скрипт, выводящий на экран целые числа в заданном диапазоне с использованием оператора while**

```
1. #!/bin/sh
#вывода на экран целых чисел от 1 до 20.
2. # Count from 1 to 20
3. i=1
4. while [ $I -le 20 ]
5. do
6.         echo $I
7.        i= $I + 1
8. done
9. exit 0
```

В этом примере в строке 3 переменной i присваивается начальное значение 1, i применяется как счетчик цикла, поэтому здесь нет необходимости использовать описательное имя. Обращаю внимание, что имена переменных в разных регистрах имеют разное значение, кроме того, присвоение переменной сложного значения оформляется как составная переменная!

Строка 4: Команда while содержит условие, заключенное в квадратные скобки. На самом деле, они представляют собой сокращенную запись команды test, которая часто используется в сценариях командного интерпретатора. Команда test использует достаточно прозрачный синтаксис, -1e в данном примере обозначает "меньше или равно". Таким образом, цикл выполняется до тех пор, пока значение переменной i меньше или равно 20.

Строка 5: Оператор do указывает, что все операторы, следующие за ним, должны выполняться при каждой итерации цикла. Все выражения между do и done являются телом цикла.

Строка 6: Здесь выводится значение переменной i.

Строка 7: Здесь используется подстановка команд, значение переменной i увеличивается на единицу, а затем новое значение вновь присваивается переменной.

Строка 8: Завершающий оператор цикла. В этой точке программа вновь возвращается к оператору while и вновь проверяет условие цикла. Если значение переменной i все еще меньше или равно 20, операторы в теле цикла выполняются снова. Если i больше 20, цикл завершается и управление передается оператору, следующему за оператором done (в данном случае, это просто оператор exit в строке 9, который завершает программу с кодом успешного выполнения, равным 0).

Операторы внутри цикла выровнены с отступом по левому краю. Это позволяет легко выделить цикл при чтении исходного кода. Интерпретатор игнорирует отступ, выполняя команды обычным образом. Пробел между [и

тестируемым условием является обязательным. Его отсутствие приведет к ошибке. Например, [*$I -le 20*] выполняется, а [*$I -le 20*] возвращает ошибку.

9) **Перехват прерываний**.

```
#!/bin/sh
# Программа демонстрирует перехват прерываний
trap 'echo "Interrupt received. Quitting." 1>&2' 1 2 3 15
echo -n "Enter a number: "
readln num
exit 0
```

10) **Пример функции, удаляющей временные файлы при завершении работы сценария.**

```
#!/bin/sh
on_exit() {
        rm -rf /tmp/myprogram.*
        mv logfile logfile.old
        mail student@evm-bi.com < report.txt
}
```

Пример вызова данной функции:

```
trap on_exit 0 1 2 3 15
```

4.6. Вопросы к защите лабораторной работы

1. Назначение и возможности языка командного интерпретатора Bourne Shell.
2. Что такое сценарий и для чего он предназначен?
3. Перечислите базовые классы и типы прав доступа к файлу.
4. Какие подстановки выполняет командный интерпретатор?
5. Поясните использование условных операторов в скриптах.
6. Поясните использование операторов организации цикла в скриптах.
7. Укажите различные форматы команды организации циклических операций.
8. Что понимается под shell-функциями?
9. Поясните структуру функции.
10. Поясните принцип использования функций в сценариях.
11. Как выполнить отладку сценария командного интерпретатора?
12. Поясните на примере, как выполняется перехват прерываний?

4.7. Список рекомендуемой литературы

1) Романчева Н.И. ОС Linux – принципы, технологии, инструменты / Учебное пособие.- М.: МГТУ ГА, 2011.

2) Робачевский А. Операционная система UNIX.-СПб.:BHV-Санкт-Петербург, 2020- 528 с.