МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФИЛИАЛ ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ « МЭИ» в г. СМОЛЕНСКЕ

Я.А. Федулов, А.С. Войцицкая

ОСНОВЫ РАБОТЫ И ПРОГРАММИРОВАНИЯ В ОПЕРАЦИОННОЙ СИСТЕМЕ LINUX

Методические указания по изучению курса «Программное обеспечение автоматизированных систем»

Смоленск
2019

УДК 004.056 (076.5)

М -18

Допущено учебно-методическим Советом филиала ФГБОУ ВО «Национальный исследовательский университет «НИУ «МЭИ» в г. Смоленске в качестве методического пособия для студентов, обучающихся по направлению подготовки магистров 10.04.01 «Информационная безопасность».

Подготовлено на кафедре «Вычислительной техники»

Рецензент
кандидат технических наук …

Федулов Я.А. Основы работы и программирования в операционной системе LINUX/, Я.А. Федулов. – Смоленск: филиал ФГБОУ ВО «Национальный исследовательский университет «МЭИ» в г.Смоленске, 2019. - …… с.

В настоящем методическом пособии рассматриваются вопросы разработки программного обеспечения на операционной системе LINUX. Данная система является свободно распространяемой и бесплатной, для дальнейшей работы в области IT необходимо знать и владеть навыками работы в данной ОС.

Предназначено для студентов, обучающихся по направлению подготовки магистров 10.04.01 «Информационная безопасность».

ВВЕДЕНИЕ

Данные методические указания разработаны в соответствии с учебно-методическим комплексом по дисциплинам «Программирование» и «Технология объектного - программирования» для студентов специальности «Программное обеспечение средств вычислительной техники и автоматизированных систем».

В методических указаниях рассматриваются следующие вопросы: разработки программного обеспечения на операционной системе LINUX, основные функции для работы в данной ОС, использование стандартных приемов проектирования при разработке программного обеспечения.

В первой лабораторной работе предлагается изучить оболочку bash и приобрести навыки работы с основными командами.

1. ЛАБОРАТОРНАЯ РАБОТА № 1. ОСНОВЫ РАБОТЫ В КОМАНДНОЙ ОБОЛОЧКЕ BASH

Цель работы: изучение командной оболочки bash и приобретение навыков работы с ее основными командами.

1.1 Теоретическое введение

Командная оболочка bash

Bash — это наиболее распространенный командный интерпретатор в операционных системах семейства Linux, с его помощью пользователь может либо вводить команды операционной системе по отдельности, либо запускать скрипты, состоящие из списка доступных команд.

Работа с командной оболочкой **bash** осуществляется в консоли, доступ к которой можно получить с помощью виртуального терминала или эмулятора терминала. Для перехода в виртуальный терминал следует нажать комбинацию клавиш **Ctrl+Alt+F*n*** (вместо *i* могут быть числа от 1 до 7, ***n*** = 7 для возврата в графический терминал), запуск эмулятора терминала выполняется в графической оболочке через главное меню Приложения→Стандартные→Терминал. В результате на экране будет отображаться однотонный текстовый экран (в случае эмулятора терминала — часть экрана), вверху которого будет выведено приглашение вида student@localhost:~$. По виду окончания приглашения можно определить, является ли текущий пользователь суперпользователем ***root*** («#» — для root и «**$**» — для остальных пользователей). Все манипуляции в консоли совершаются с помощью команд, набранных с клавиатуры.

Поиск нужной команды можно осуществить, набрав предполагаемое начало команды и нажав дважды клавишу **Tab**, в результате чего будет получен список команд, начинающийся с введенных символов.

Для каждой доступной команды можно получить описание (справку). Рекомендуется вызывать его всякий раз перед первым вызовом команды. Справка вызывается командой вида:

```
man <имя команды>
```

либо

<имя команды> --help

У команды **man** также имеются дополнительные опции, о которых можно узнать, набрав в командной строке «**man man**» или «**man --help**». Выход из режима чтения справки **man** происходит по нажатию клавиши q.

При работе в консоли на экране умещается ограниченное количество строк, для просмотра остальных строк необходимо воспользоваться прокруткой с помощью сочетания клавиш **Shift+PageUp** (прокрутка вверх) и **Shift+PageDown** (прокрутка вниз).

В течение каждого сеанса работы с **bash** в файле **.bash_history** сохраняется история команд (имя файла установлено в переменной *$HISTFILE*). Файл находится в домашнем каталоге пользователя, их вызвавшего. Количество команд, записываемых в файл истории, ограничено и установлено в переменной *$HISTSIZE*. Значение этих и других системных переменных можно узнать командой

echo $HISTFILE $HISTSIZE

Для просмотра истории команд используется команда **history**. Для просмотра определенного количества команд, необходимо указать их число в качестве аргумента: **history 5**

Перемещение между предыдущими командами осуществляется нажатием клавиш «вверх» и «вниз». Прокрутка содержимого терминала осуществляетсянажатием клавиш **Shift+PageUp** (прокрутка вверх) и **Shift+PageDown** (прокрутка вниз).

1.2 Подготовка к работе

1. Изучите команды работы с файлами: создание **touch**, перезапись файла **echo** "текст" > file.txt, запись в конец файла **echo** "текст" >> file.txt, просмотр **cat**, **page**, **more**, **less**, редактирование **nano**, копирование **cp**, перемещение/переименование **mv**, удаление **rm**, **shred,** сравнения **diff**.
2. Изучите команды работы с каталогами: создание **mkdir**, просмотр **ls**, переход между каталогами **cd**, удаление **rmdir**.
3. Изучите средства перенаправления ввода-вывода в **bash** и способы использования специальных файлов устройств терминалов /dev/tty.
4. Изучите способы получения справки по использованию команд.
5. Изучить команду записи действий пользователя **script**.

1.3 Задание для самостоятельной работы

1. Очистите экран (команда **clear**).
2. Начать запись текущего сеанса в файл <фамилия>.txt.
2. В домашнем каталоге пользователя создайте одной командой заданную структуру: каталоги main_*n*, A*n*, B*n*, C*n*, D*n*, где n — номер варианта по журналу (рисунок 1).

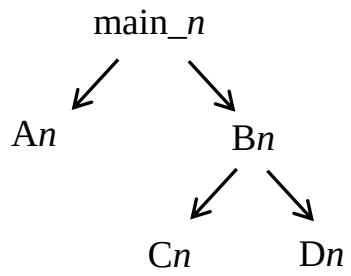


Рисунок 1 — Структура каталогов

3. В каталоге main_*n* создайте текстовый файл name.txt со своей фамилией с помощью команды **echo** и средств перенаправления ввода-вывода.
4. Допишите в файл name.txt из каталога main_*n* название группы с помощью команды **echo** и средств перенаправления ввода-вывода.
5. В каталоге A*n* создайте текстовый файл date.txt с текущем временем в формате ДД.ММ.ГГГГ ЧЧ:ММ:СС (например, 12.10.2018 14:15:19), используя команду **date** и средства перенаправления ввода-вывода (обратите внимание на символ «+» перед описанием пользовательского формата даты).
6. Скопируйте файлы name.txt и date.txt в каталоги B*n* и C*n* соответственно.
7. Файл name.txt в каталоге main_*n* переименуйте в myname.txt.
8. Файл date.txt из каталога C*n* переместите в каталог D*n*.
9. Введите с клавиатуры файл list.txt со списком фамилий одногруппников (5–6 фамилий) путем копирования из специального файла устройства терминала /dev/tty.
10. Отредактировать текстовым редактором **nano** файл myname.txt, добавив в него свой номер варианта.
11. Выведите на экран содержимое файлов name.txt из каталога B*n* и date.txt из каталога C*n*.
12. На место файла name.txt из каталога B*n* скопируйте файл date.txt из каталога C*n*.
13. На место файла date.txt из каталога C*n* скопируйте файл myname.txt из каталога main_*n*.
14. Выведите на экран содержимое файлов name.txt из каталога B*n* и date.txt из каталога C*n*.
15. Удалите файл name.txt из каталога B*n*.
16. Выведите список всех созданных файлов и каталогов с помощью одной команды.
17. Сохраните историю команд в файл history.txt в каталоге D*n*.
18. Очистите экран и покажите созданные структуры и содержимое файлов преподавателю.
19. Удалите все созданные структуры одной командой.
20. Остановите запись сеанса.

1.4 Контрольные вопросы

2.1 Как получить справку по функции printf языка программирования C?

2.2 Приведите практические примеры использования средств перенаправления ввода-вывода.

2.3 Чем отличаются команды вывода содержимого файлов на экран **page**, **less** и **more**?

2.4 Объедините два текстовых файла в один с помощью команды **cat**.

2.5 Как организовать чат между пользователями двух терминалов?

2.6 Что такое переменная окружения?

2.7 Как определить полное имя домашнего каталога текущего пользователя?

2.8 Приведите все способы записи пути к файлу myname.txt (текущий каталог — D*n*).

2.9 Как скопировать файл без использования команды **cp**?

2.10 Как создать скрытый каталог?

2. ЛАБОРАТОРНАЯ РАБОТА № 2. АДМИНИСТРИРОВАНИЕ ПОЛЬЗОВАТЕЛЕЙ В LINUX

Цель работы: изучение способов администрирования пользователей в ОС Linux.

2.1 Теория

2.1.1 Работа с пользователями

ОС Linux — многопользовательская система, на одном компьютере может быть несколько различных пользователей, каждый из которых имеет свои собственные настройки, данные и права доступа к файлам и командам.

Для упрощения администрирования большого количества пользователей в ОС Linux введены **группы**. Любой файл и каталог в ОС Linux имеет **пользователя-владельца** и **группу-владельца**. Каждая группа, также как и отдельный пользователь, обладает набором прав доступа к различным компонентам системы. Каждый пользователь — член этой группы — автоматически получает все права группы. Таким образом, группы служат для группировки пользователей по принципу одинаковых полномочий на какие-либо действия. Каждый пользователь может состоять в неограниченном количестве групп и в каждой группе может быть сколько угодно пользователей.

При работе с пользователями возможны следующие манипуляции:

- добавление пользователя (useradd) / группы (groupadd);
- удаление пользователя (userdel) / группы (groupdel);
- модификация настроек пользователя (usermod) и пароля пользователя (passwd) / группы (groupmod, gpasswd);
- просмотр информации о пользователе / группе (id).

Для выполнения этих команд необходимы права суперпользователя **root**, получить которые можно с помощью команд **su** или **sudo**. Команда **su** изменяется пользователя-владельца текущей сессии терминала, после этого все действия выполняются от имени нового пользователя (как правило, суперпользователя **root**). При этом действия пользователя не ограничены и не контролируемы. Команда **sudo** позволяет выполнить команду от имени суперпользователя (или другого пользователя) при наличии делегированных ему полномочий, описанных в файле */etc/sudoers*. При этом для каждого пользователя ведется журнал команд, выполненных от имени суперпользователя **root**.

Пример использования команд работы с пользователями:

```
sudo groupadd test_group
sudo useradd -g test_group -s /bin/bash test
sudo passwd test
```

Последняя команда требует следующего диалога с пользователем:

Enter new UNIX password:<ввод пароля не отображается>
Retype new UNIX password:<ввод пароля не отображается>
passwd: password updated successfully

Проверить результат можно командой **id <имя пользователя>:**

id test
uid=1001(test) gid=1002(test_group) groups=1002(test_group)

В представленном выше примере добавляется группа **test_group** для нового пользователя (**groupadd**), далее создается новый пользователь, имеющий основную группу **test_group**, логин **test** и использующий командную оболочку **bash**, далее для пользователя **test** устанавливается пароль (**passwd test**), после чего проверяются параметры созданного пользователя (**id test**). В выводе команды **id** видно, что UID (user identifier) и GID (group identifier) — более 1000. Это является признаком **обычного пользователя**. Значения меньше 1000 (а в некоторых дистрибутивах — меньше 500) указывают на то, что пользователь является **системным пользователем.** Обычно **системные пользователи** имеют id меньше, чем 100, а **суперпользователь root** имеет id, равный 0. Автоматическая нумерация обычных пользователей начинается со **значения UID_MIN**, установленного в файле */etc/login.defs*, это значение обычно установлено в 500 или 1000.

Помимо учетных записей **обычных пользователей** и учетной записи суперпользователя **root**, обычно в системе бывает несколько учетных записей специального назначения для служб (в ОС Linux называются демонами), таких как FTP, SSH, mail, news и т.д. Такие учетные записи часто управляют файлами, но к ним невозможно получить доступ путем обычной регистрации в систем. Поэтому обычно они имеют *login shell*, определенный как **/sbin/nologin** или **/bin/false**, чтобы попытки зарегистрироваться в системе терпели неудачу.

В некоторых дистрибутивах команды для работы с пользователями имеют расширенные функции. Например, команда **useradd** по умолчанию, для нового пользователя создает новую группу и для отмены данной функции, необходимо использовать опцию **-N** или явно указать группу по ключу **-g**. Для уточнения таких вопросов необходимо обратиться к документации по используемой команде (**man <команда>** или **<команда> -- help**).

Список всех групп доступен в файле */etc/group*, который можно вывести на экран с помощью команды **cat**.

Список некоторых утилит для администрирования пользователей и групп:

- **last** — показывает, какие пользователи последними входили в систему (и покидали ее), выполняя для этого поиск в файле /var/log/wtmp в обратном порядке; также показывает информацию о загрузках системы, ее остановках и изменениях ее уровней запуска;
- **lastb** — показывает неудачные попытки входа в систему, которые записаны в файле /var/log/btmp;

- **w** — показывает активных пользователей;
- **whoami** — имя текущего пользователя;.
- **finger user@host** — показать информацию о user (без указания пользователя выведет список активных пользователей) на хосте host, утилита не чувствительна к регистру символов;
- **write user [terminal]** — начать сеанс общения с пользователем user на терминале terminal
- **talk user@host** — чат с пользователем user на компьютере host;
- **wall message** — разместить сообщение message на всех терминалах;
- **mesg y/n** — включение (y) и отключение (n) возможности принимать сообщения на консоли;
- **su user** — создание оболочки (подоболочки текущей оболочки) с правами пользователя user (без указания пользователя - вызывается оболочка root);
- **useradd user** — добавление нового регистрационного имени пользователя user в системе;
- **usermod user** — изменение настроек пользователя с именем user в системе;
- **userdel user** — удаляет пользователя user из системы
- **users** — имена пользователей, зарегистрированных в системе;
- **groupadd group** — добавление (создание) новой группы group в системе;
- **groupmod group** — изменение информации о группе group в системе;
- **groupdel group** — удаление группы group из системы;
- **passwd user** — изменяет/устанавливает пароль пользователя user;
- **gpasswd group** — изменяет/устанавливает пароль группы group (наличие пароля группы позволяет пользователям временно войти в группу при помощи команды newgrp, если им известен пароль группы);
- **chage** — установка срока действия пароля пользователя;
- **id who** — просмотр информации о пользователе/группе who (принадлежность к группам, UID, GID);
- **ulimit** — ограничение пользовательских ресурсов.

Подробности использования и опции команд можно получить в справке по команде **man**.

2.1.2 Права доступа

У любого файла и каталога (а любой каталог является файлом) есть три группы прав доступа: группа прав пользователя-владельца, группа прав членов группы-владельца и группа прав всех остальных пользователей системы. Каждая группа состоит из прав на чтение, запись и запуск файла на исполнение.

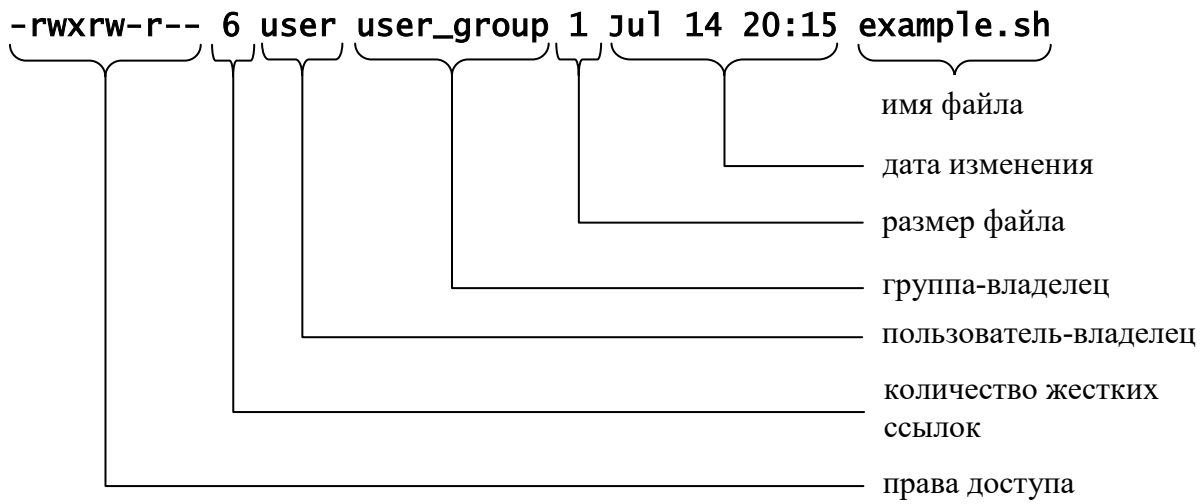
Права доступа для каталогов имеют следующие особенности:

- чтение (**r**ead) — получение списка имен файлов каталога;
- запись (**w**rite) — создание и удаление файлов (права на доступ к файлу при удалении не учитываются!);
- исполнение (e**x**ecute) — возможность перехода в каталог (право на исполнение должно быть и у всех вышестоящих каталогов).

Все права доступа действуют независимо друг от друга, например, при наличии права на исполнение и отсутствии права на чтение получается так называемый «тёмный» каталог, в котором возможен доступ к содержимому файлов, но невозможно узнать имена файлов.

Для просмотра текущего состояния прав доступа к файлу или каталога можно использовать команду:
ls –l <имя файла>.

В выводе команды для файла **ls –l example.sh**
-rwxrw-r-- 6 user user_group 1 Jul 14 20:15 example.sh



рассмотрим первую часть **–rwxrw-r--:**
«-» **–** определяет тип файла, в данном случае, обычный файл. На этом месте может стоять:
d — каталог;
b — файл блочного устройства;
c — файл символьного устройства;
s — доменный сокет (socket);
p — именованный канал (pipe);
l — символическая ссылка (link).

Первая триада «**rwx**» — права доступа для пользователя-владельца, **r** – разрешено чтение, **w** – разрешена запись, **x** – разрешено исполнение.

Вторая триада «**rw–**» — права доступа для группы-владельца на чтение и запись, **дефис** означает запрет на исполнение для членов группы-владельца.

Третья триада «**r—**» — права доступа для всех остальных: пользователи, не являющиеся владельцами файла и не входящие в группу-владельца файла, могут только прочитать файл.

Подобную информацию для каталога можно получить либо явно для каталога, либо найдя нужный каталог в выводе для каталога уровнем выше:

ls –la <имя каталога>

в выводе команды для каталога **ls –la Documents:**

total 8

drwxr-xr-x 2 user user 34002 Jul 25 19:02 .

drwxr-xr-x 27 user user 34002 Jul 25 18:14 ..

здесь **total 8** указывает общее число блоков, занимаемых на диске файлами данного каталога, а права доступа для каталога **Documents** указаны в первой строке «**drwxr-xr-x 2 user user 34002 Jul 25 19:02 .**», вторая строка относится к каталогу уровнем выше.

Изменять права доступа можно командой **chmod**, она доступна только пользователю-владельцу и суперпользователю. Ее можно использовать в двух вариантах. В первом варианте нужно явно указать, кому какое право дается или кто этого права лишается:

chmod <параметры> <имя файла>

<параметры> — это три символа, первый может принимать следующие значения и обозначает того, чьи права изменяют:

u — пользователь-владелец;

g — группа-владелец;

o — все пользователи, не входящие в группу, которой принадлежит данный файл;

a — все пользователи системы;

второй символ параметров обозначает действие над правами:

\+ — предоставляется право;

\- — лишается соответствующего права;

= — установить указанные права вместо имеющихся;

третий символ обозначает соответствующее право:

r — чтение;

w — запись;

x — выполнение.

Примеры использования команды **chmod**:

chmod a+x file_name

предоставляет всем пользователям системы право на выполнение данного файла;

chmod go-rw file_name

удаляет право на чтение и запись для всех, кроме владельца файла;

chmod ugo+rwx file_name

дает всем права на чтение, запись и выполнение.

Если опустить указание на то, кому предоставляется данное право, то подразумевается, что речь идет обо всех пользователях, т. е. вместо

chmod a+x file_name

можно записать

chmod +x file_name

Второй вариант задания команды **chmod** основан на числовом представлении прав в восьмеричном коде. Для этого символ **r** кодируется числом 4, символ **w** — числом 2, а символ **x** — числом 1. Чтобы предоставить пользователям какой-то набор прав, нужно сложить соответствующие числа. Получив, таким образом, нужные цифровые значения для владельца файла, для группы файла и для всех остальных пользователей, задаем эти три цифры в качестве аргумента команды **chmod** (ставим эти цифры после имени команды перед аргументом, который задает имя файла). Например, если нужно дать все права владельцу (4+2+1=7), право на чтение и запись — группе (4+2=6), и не давать никаких прав остальным, то команда будет выглядеть так:

chmod 760 file_name

Такое кодирование прав объясняется тем, что это восьмеричная запись тех самых 9 бит, которые задают права для владельца файла, группы файла и для всех пользователей.

Для того, чтобы иметь возможность изменить права группы, владелец должен дополнительно быть членом той группы, которой он хочет дать права на данный файл.

Существует еще три возможных атрибута файла, устанавливаемых с помощью той же команды **chmod**. Это атрибуты для исполняемых файлов, которые в индексном дескрипторе файла в двухбайтовой структуре, определяющей права на файл, занимают позиции 5-7, сразу после кода типа файла.

Первый из этих атрибутов — так называемый «бит смены идентификатора пользователя». Смысл этого бита состоит в следующем.

Обычно, когда пользователь запускает некоторую программу на выполнение, эта программа получает те же права доступа к файлам и каталогам, которые имеет пользователь, запустивший программу. Если же установлен «бит смены идентификатора пользователя», то программа получит права доступа к файлам и каталогам, которые имеет владелец файла программы. Это позволяет решать некоторые задачи, которые иначе было бы трудно выполнить. Самый характерный пример — команда смены пароля **passwd**. Вся информация о пользователях хранится в файле */etc/passwd*, а зашифрованные пароли — в файле */etc/shadow*, владельцем которых является суперпользователь **root**. Поэтому программы, запущенные обычными пользователями, в том числе команда **passwd**, не могут производить запись в эти файлы. А, значит, пользователь как бы не может изменить свой собственный пароль. Но для файла */usr/bin/passwd* установлен «бит смены идентификатора владельца», каковым является пользователь **root**. Следовательно, программа смены

пароля **passwd** запускается с правами **root** и получает право записи в файлы */etc/passwd* и */etc/shadow* (уже средствами самой программы обеспечивается то, что пользователь может изменить только свою строку в этих файлах).

Установить «бит смены идентификатора владельца» может суперпользователь с помощью команды

chmod +s file_name

Аналогичным образом работает «бит смены идентификатора группы».

Еще один возможный атрибут исполняемого файла — это «бит сохранения задачи» или «sticky bit» (дословно — «бит прилипчивости»). Этот бит указывает системе, что после завершения программы необходимо сохранить ее в оперативной памяти. Удобно включить этот бит для задач, которые часто вызываются на исполнение, так как в этом случае экономится время на загрузку программы при каждом новом запуске. Этот атрибут был необходим на старых моделях компьютеров. Сейчас он используется редко и применяется только для каталогов, чтобы защитить в них файлы. Из такого каталога пользователь может удалить только те файлы, владельцем которых он является. Примером может служить каталог */tmp*, в котором запись открыта для всех пользователей, но нежелательно удаление чужих файлов.

Если используется цифровой вариант задания атрибутов в команде chmod, то цифровое значение этих атрибутов должно предшествовать цифрам, задающим права пользователя:

chmod 4775 file_name

При этом веса этих битов для получения нужного суммарного результата задаются следующим образом:

4 — бит смены идентификатора пользователя (setuid),

2 — бит смены идентификатора группы (setgid),

1 — бит сохранения задачи (sticky bit).

Если какие-то из этих трех битов установлены в 1, то несколько изменяется вывод команды **ls -l** в части отображения установленных атрибутов прав доступа. Если установлен «бит смены идентификатора пользователя», то символ «**x**» в группе, определяющей права владельца файла, заменяется символом «**s**». Причем, если владелец имеет право на выполнение файла, то символ «**x**» заменяется на «**s**», а если владелец не имеет права на выполнение файла (например, файл не исполняемый), то вместо «**x**» ставится «**S**». Аналогичные замены имеют место при задании «бита смены идентификатора группы». Если равен 1 «бит сохранения задачи (sticky bit)», то заменяется символ «**x**» в группе атрибутов, определяющей права для всех остальных пользователей, причем «**x**» заменяется символом «**t**», если все пользователи могут запускать файл на выполнение, и символом «**T**», если они такого права не имеют. Таким образом, хотя в выводе команды ls -l не предусмотрено отдельных позиций для отображения значений битов смены идентификаторов и бита сохранения задачи, соответствующая информация выводится. Пример того, как это будет выглядеть:

ls -l prim1
-rwSrwsrwT 1 ck root 12 Jan 25 23:17 prim1

Кроме изменения самих прав доступа можно изменять пользователя-владельца файла командой **chown** и группу-владельца командой **chgrp.**

2.2 Подготовка к работе

1. Изучите команды администрирования пользователей в Linux.
2. Изучите команды и способы настройки доступа к файлам и каталогам.
3. Исследуйте возможность выполнения операций с файлами при изменении прав доступа к каталогу, в котором они находятся.
4. Изучите форматы файлов /etc/group, /etc/passwd, /etc/shadow.

2.3 Задание для самостоятельной работы

1. Все действия выполняйте исключительно с помощью клавиатуры.
2. Создайте новую группу с именем вашей группы (например, PE1_15).
3. Создайте нового пользователя с вашей фамилией, входящего в созданную в п. 2 группу, группы student и sudo. При создании пользователя выберите оболочку */bin/bash*, создайте свой домашний каталог и укажите в примечании свое полное имя (опциями команды **useradd**). Установите пароль для входа в систему.
4. Войдите в терминал tty1 созданным в п. 3 пользователем.
5. Создайте три новых пользователя для студентов группы. Для всех пользователей создайте свой домашний каталог, выберите оболочку */bin/bash*, указав в примечании полное имя пользователя. Каждый пользователь должен входить в группу, созданную в п. 2, один пользователь должен также входить в группу student. Для каждого пользователя установите свой пароль.
6. Переименуйте одного из пользователей <имя_пользователя> в <имя_пользователя>_ (добавьте в конец символ подчеркивания _) и установите для него командную оболочку /sbin/nologin.
7. Соберите информацию обо всех новых пользователях в файле users.txt.
8. Дополните файл users.txt информацией о последних входах в систему (см. **last**).
9. Создайте каталог /home/shared, доступный всем пользователям группы, созданной в п. 2.
10. Войдите в систему всеми новыми пользователями и пользователем student (в терминалах tty2–tty4).
11. Получите список активных пользователей (см. **w**).

12. Создайте в каталоге /home/shared текстовые файлы file_<имя пользователя>_1.txt, file_<имя пользователя>_2.txt с правами 640 и 604 соответственно для каждого пользователя. Запишите в файлы их имена.
13. Убедитесь, в наличии или отсутствии возможности скопировать файлы из п.12 в свою домашнюю папку всеми пользователями (без команд **su** или **sudo**).
14. Попытайтесь удалить один из файлов, недоступных для чтения в п. 13. Проанализируйте результат.
15. Измените группу-владельца всех файлов на student.
16. Проделайте п.13 пользователем student. Проанализируйте результат.
17. Скопируйте в свой домашний каталог файл */bin/ls*, запретите его исполнение. Попытайтесь запустить на исполнение этот файл из домашнего каталога и проанализируйте результат.
18. Попытайтесь просмотреть содержимое домашнего каталога суперпользователя */root*. Предварительно создайте в каталоге */root* текстовый файл.
19. Установите файлу *~/ls* (см. п. 17) пользователя-владельца суперпользователя root и бит смены идентификатора пользователя, и с его помощью проделайте п. 18. Проанализируйте результат.
20. Запишите историю команд занятия в файл.
21. Покажите результаты работы преподавателю.
22. Удалите все созданные файлы и каталоги, всех созданных пользователей и группу.

2.4 Контрольные вопросы

1. На какие операции с файлами оказывают влияние права доступа к файлу, а на какие — права доступа к каталогу, в котором содержится указанный файл?
2. Можно ли отредактировать файлы с правами доступа 204 и 240 и каким образом?
3. Пользователь установил права доступа к файлу только на чтение — 444, но хакеру удалось отредактировать этот файл. Какие ошибки мог совершить пользователь?
4. Как разрешить пользователю удалять из каталога только те файлы, владельцем которых он является?
5. Что такое тёмный каталог?
6. Как определить только с помощью команды **cp** в какие группы входит пользователь student?
7. Как суперпользователю root запретить всем пользователям самостоятельно изменять пароль?
8. Как запретить вход в систему суперпользователю root?
9. Как получить список всех пользователей системы?
10. В чем разница между командами **su** и **sudo**?

3. ЛАБОРАТОРНАЯ РАБОТА № 3. ЖЕСТКИЕ И СИМВОЛИЧЕСКИЕ ССЫЛКИ, ПРАВА ДОСТУПА

Цель работы: изучение особого типа файлов — ссылок.

3.1 Теория

В ОС Linux вся информация о файле привязана не к имени файла, а так называемому **индексному дескриптору**. У каждого файла есть свой уникальный (единственный и неповторимый) индексный дескриптор, который содержит сведения о файле: в каких блоках диска хранится содержимое файла, размер файла, время его создания и др.

Пронумерованные индексные дескрипторы файлов содержатся в специальной таблице. Каждый логический и физический диск имеет собственную таблицу индексных дескрипторов. Именно номер индексного дескриптора является истинным именем файла в системе**.**

Поскольку пользователю работать с осмысленными словами куда удобнее, чем с огромными числами дескрипторов и их номеров, поэтому любому файлу в системе обычно дается осмысленное имя (обычно словесное), которое не содержит информации о файле, а лишь указывает (ссылается) на его дескриптор.

Имя файла, ссылающееся на его индексный дескриптор, называется **жесткой ссылкой**. Механизм жестких ссылок — это основной способ обращения к файлу по имени в ОС Linux.

У файла может быть несколько жёстких ссылок: в таком случае он будет фигурировать на диске одновременно в различных каталогах и/или под различными именами.

Количество жёстких ссылок файла сохраняется на уровне файловой системы в метаинформации и можно узнать командой:

ls –l <имя файла>

в выводе команды

-rwxrwxrwx 6 user user 34002 Jul 14 20:15 example.txt

число 6 — это количество жестких ссылок на файл **example.txt**.

Файлы с нулевым количеством ссылок перестают существовать для системы и, со временем, будут перезаписаны физически. При создании файла на него автоматически создаётся одна жёсткая ссылка (на то место файловой системы, в котором файл создаётся). Дополнительную ссылку можно создать с помощью команды **ln:**

ln <путь и имя файла-источника> <путь и имя файла-назначения>

путь может быть полным или относительным.

Все ссылки одного файла равноправны и неотличимы друг от друга — нельзя сказать, что файл существует в таком-то каталоге, а в других местах есть лишь их копии. Удаление любой из ссылок приводит к удалению файла лишь в том случае, когда удалены все остальные жёсткие ссылки на него.

В связи с тем, что жёсткие ссылки ссылаются на индексный дескриптор, уникальный в пределах дискового раздела, создание жёсткой ссылки на файл в каталоге другого раздела невозможно. Для преодоления этого ограничения используются символьные ссылки.

Символические ссылки — эти ссылки тоже могут рассматриваться как дополнительные имена файлов, но в то же время они представляются отдельными файлами — файлами типа символических ссылок. В отличие от жёстких ссылок, символические ссылки могут указывать на файлы, расположенные в другой файловой системе, например, на монтируемом носителе, или даже на другом компьютере. Если исходный файл удален, символическая ссылка не удаляется, но становится бесполезной. Используйте символические ссылки в тех случаях, когда хотите избежать путаницы, связанной с применением жестких ссылок.

Создание любой ссылки внешне подобно копированию файла, но фактически как исходное имя файла, так и ссылка указывают на один и тот же реальный файл на диске. Поэтому, например, при внесении изменения в файл, обратившись к нему под одним именем, вы обнаружите эти изменения и тогда, когда обратитесь к файлу по имени-ссылке. Для того, чтобы создать символическую ссылку, используется уже команда ln с дополнительной опцией -s:

ln –s <путь и имя файла-источника> <путь и имя файла-назначения>

Пример:

создадим символическую ссылку:

ln –s /home/user/example.txt /home/user/template.txt

выведем сведения о файле-ссылке:

ls –l /home/user/template.txt

в выводе команды ls

lrwxrwxrwx 1 user user 34002 Jul 14 20:15 template.txt -> /home/user/example.txt

самый первый символ в этой строке «l» показывает, что данная запись соответствует символической ссылке. Также это видно и в поле имени, где после нового имени и стрелки указано исходное имя файла.

Удаляются жесткие и символические ссылки командами:

rm <имя файла-ссылки> - для жестких и символических ссылок;

unlink <имя файла-ссылки> - для символических ссылок.

Различие между копированием и созданием ссылок в том, что с помощью ссылок для одного и того же файла можно задать несколько имен, тогда как при копировании создается два объекта с идентичными данными, но с разными именами. Несомненно, следует использовать копирование для со-

здания резервных копий, а также при тестировании новых программ, чтобы не подвергать риску рабочие данные. Ссылки имеет смысл использовать тогда, когда необходимо создать псевдоним для файла (или директории), возможно, более удобный в использовании.

При обновлении файла обновляются и все ссылки на него, чего не происходит в случае его копирования. Также символические ссылки могут оказаться «битыми», но в результате последующих операций записи может быть создан новый файл.

3.2 Подготовка к работе

1. Изучить понятия жесткой и символической ссылок и способы их создания (см. руководство по команде ln: **man ln**).
2. Исследовать свойства файлов-ссылок.

3.3 Задание для самостоятельной работы

1. Создайте в домашнем каталоге каталог A1 с файлом name.txt со своим именем.
2. Создайте в домашнем каталоге каталог B1 с файлом class.txt с именем группы.
3. Создайте внутри каталога B1 каталог B2.
4. В каталоге B2 создать символическую ссылку class_sym.txt на файл class.txt из каталога B1.
5. Отредактировать текстовым редактором **nano** файл по символической ссылке class_sym.txt, добавив в него свой номер варианта.
6. Вывести на экран содержимое файла class.txt из каталога B1.
7. Вывести содержимое символической ссылки class1_sym.txt.
8. На место файла class.txt из каталога B1 скопировать файл name.txt из каталога A1.
9. Вывести содержимое символической ссылки class_sym.txt.
10. В каталоге B3 (каталог B3 создать в домашнем каталоге) создать жесткую ссылку name_h.txt на файл name.txt из каталога A1.
11. Отредактировать файл по ссылке name_h.txt, вставив в его начале фразу “Текущая дата:” и добавив время (пример: “Текущая дата: 12.10.2019”).
12. Просмотреть содержимое файла по ссылке name_h.txt.
13. место файла name.txt из каталога A3 скопировать файл class_sym.txt из каталога A1.
14. Просмотреть содержимое файла по ссылке name_h.txt.

15. Удалить файл name.txt из каталога A3.
16. Просмотреть содержимое файла по ссылке name_h.txt.
17. Добавьте любой текст в конец файла по ссылке name_h.txt.
18. Вывести содержимое всех каталогов с помощью одной команды.
19. Очистить экран и показать созданные структуры и содержимое файлов преподавателю.
20. Удалить все созданные структуры одной командой.

3.4 Контрольные вопросы

1. Что называется жесткими ссылками?
2. Какие свойства файлов-ссылок?
3. Что представляют символические ссылоки?
4. Какие существуют способы созданияссылок?
5. Что происходит с файлами с нулевым количеством ссылок?
6. Что называется именем файла в системе?

4 ЛАБОРАТОРНАЯ РАБОТА № 4. ОСНОВЫ РАБОТЫ В ФАЙЛОВОМ МЕНЕДЖЕРЕ MIDNIGHT COMMANDER

Цель работы: приобретение навыков работы в файловом менеджере Midnight Commander.

4.1 Теория

Файловый менеджер Midnight Commander (MC) является классическим двухпанельным менеджером, аналогичным Norton Commander, FAR и т. д. Внешний вид окна файлового менеджера представлен на рис. 1.

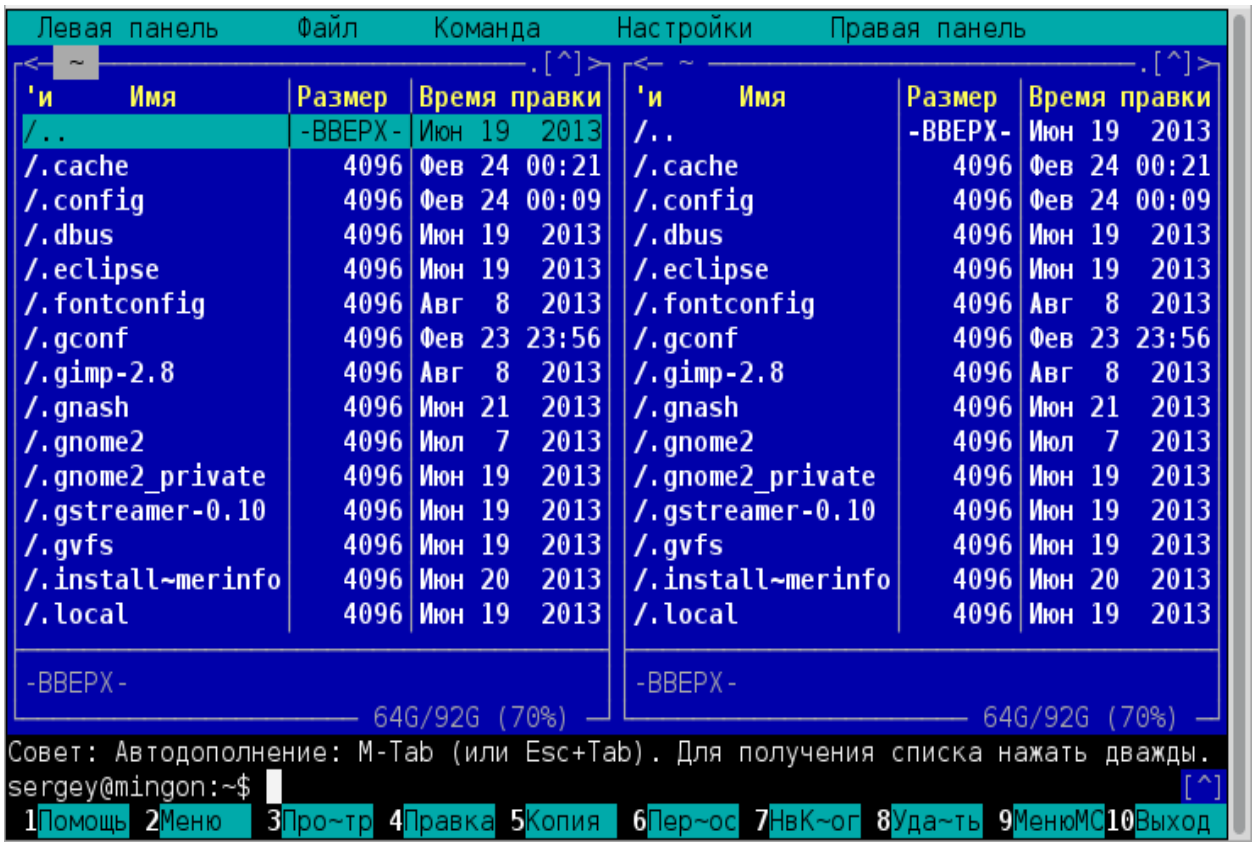


Рисунок 1 – Внешний вид Midnight Commander

Файловый менеджер запускается командой **mc** (при необходимости использовать курсор в MC следует запускать с ключом **-x**). При этом эран разделен на две панели со списками файлов и каталогов, верхнюю панель меню и нижнюю панель функциональных клавиш, командную строку с текущим интерпретатором. Для работы в файловом менеджере используются клавиша TAB для смены активной панели и стрелки вверх, вниз, клавиши Page Up и Page Down, Home и End для просмотра содержимого каталога.

Назначение функциональных клавиш нижней панели (при отсутствии клавиш F1 — F10 используется сочетание клавиш Alt + 1 — Alt + 0):
F1 — справка;
F2 — пользовательское меню (архивация файлов и т. д.);
F3 — просмотр содержимого файла;
F4 — редактирование файла (MC позволяет настроить то, какой редактор будет использоваться по умолчанию);
F5 — копирование файла;
F6 — переименование или перемещение файла;
F7 — создание каталога;
F8 — удаление каталога или файла;
F9 — меню Midnight Commander;
F10 — выход.

Часто используемые сочетания клавиш:
Shift-F4 — создать файл;
Ctrl-O — скрыть / показать панели;
Insert — выделение файла;
Ctrl-R — обновление панели;
Alt-Shift-? — поиск файла;
Ctrl-X D (отпустить Ctrl-X перед нажатием D) — сравнение каталогов;
Alt-C — смена каталога;
Shift-+ — выделение группы;
Shift-* — инверсия выделения;
\ — снятие выделения группы;
Ctrl-X L (S) — создание жесткой ссылки (или символической ссылки соответственно).
Ctrl-X C (O) — просмотр прав доступа и владельца.

4.2 Задание для практического занятия

Изучить содержание панели и меню Midnight Commander, назначение функциональных клавиш.
Создать заданную структуру каталогов, файлов и ссылок.
Изучить основные настройки Midnight Commander.
Изучить способы получения справки по использованию команд.

4.3 Задание для самостоятельной работы

Все действия выполняйте исключительно с помощью клавиатуры.
Создать структуру каталогов **Cat1**, **Cat2**, **Cat3**, **Cat4**, **Cat5**, **Cat6** в домашнем каталоге.
В каталоге **Cat1** создать текстовый файл file1.txt со своей фамилией и номером варианта.

В каталоге **Cat2** создать текстовый файл file2.txt с текущей датой в формате ДД.ММ.ГГГГ ЧЧ:ММ:СС (например, 12.10.2015 14:15:19), используя команду **date** и средства перенаправления ввода-вывода.
Скопировать файлы file1.txt и file2.txt в каталоги **Cat3** и **Cat4** соответственно.
Файл file1.txt в каталоге **Cat1** переименовать в file1_1.txt.
В каталоге **Cat5** создать символическую ссылку file1_s.txt на файл file1.txt из каталога **Cat3**.
Отредактировать файл по символической ссылке file1_s.txt, добавив в него свой номер группы.
Просмотреть содержимое файла file1.txt из каталога **Cat3**.
Удалить файл file1.txt из каталога **Cat3**, просмотреть, как изменилась ссылка file1_s.txt.
Создать жесткую ссылку file2_h.txt в каталоге **Cat6** на файл file2.txt из каталога **Cat2**.
Включить использование встроенного редактора mcedit в настройках Midnight Commander.
Отредактировать файл по ссылке file2_h.txt, вставив в его начале фразу “Текущая дата:” и удалив время (пример: “Текущая дата: 12.10.2015”). Повторить эту строку 5 раз с помощью команд копирования и вставки встроенного редактора. Переместить слово «дата» из первой строки в конец файла.
Просмотреть содержимое файла file2.txt из каталога **Cat2**.
На место файла file2.txt из каталога **Cat2** переместить файл file1_1.txt из каталога **Cat1**.
Просмотреть содержимое файла по ссылке file2_h.txt в каталоге **Cat6**.
Создать цветовую подсветку для файлов *.txt
Вывести содержимое каталогов в виде дерева.
Просмотреть содержимое всех файлов с помощью быстрого просмотра.
Найти все файлы и каталоги, содержащие в имени цифру 1.
Показать результаты работы преподавателю.
Удалить все созданные структуры одной командой.

4.4 Контрольные вопросы

1. Какое содержание панели и меню Midnight Commander
2. Назначение функциональных клавиш Midnight Commander?
3. Как создать структуру каталогов, файлов и ссылок.
4. Перечислите основные настройки Midnight Commander.
5. Какие способы получения справки по использованию команд.

5 ЛАБОРАТОРНАЯ РАБОТА № 5. УПРАВЛЕНИЕ ПРОЦЕССАМИ, СИГНАЛЫ ПРОЦЕССАМ

Цель работы: изучение команд управления процессами и сигналов управления их состоянием.

5.1 Теория

Задания и процессы

Всякая выполняющаяся в Linux программа называется *процессом*. Linux как многозадачная система характеризуется тем, что одновременно может выполняться множество процессов, принадлежащих одному или нескольким пользователям. Вывести список исполняющихся в текущее время процессов можно командой **ps**, например, следующим образом:

```
  PID TTY          TIME CMD
 6997 pts/0    00:00:00 man
 7008 pts/0    00:00:00 pager
 7163 pts/1    00:00:00 ps
```

Обратите внимание, что по умолчанию команда **ps** выводит список только тех процессов, которые принадлежат запустившему её пользователю. Чтобы посмотреть все исполняющиеся в системе процессы, нужно подать команду **ps aux или любой эквивалентный вариант (см. man ps)**. *Номера процессов* (process ID, или PID), указанные в первой колонке, являются уникальными номерами, которые система присваивает каждому работающему процессу. Вторая колонка указывает имя терминала, к которому привязан процесс. Третья колонка – общее время использования процессора. Последняя колонка, озаглавленная CMD, указывает имя работающей команды. В данном случае в списке указаны процессы, которые запустил сам пользователь. В системе работает ещё много других процессов, их полный список можно просмотреть командой **ps -aux.**

Работающий процесс также называют *заданием* (job). Понятия процесс и задание являются взаимозаменяемыми. Однако, обычно процесс называют заданием, когда имеют ввиду *управление заданием* (job control). Управление заданием — это функция командной оболочки, которая предоставляет пользователю возможность переключаться между несколькими заданиями.

В большинстве случаев пользователи запускают только одно задание — это будет та команда, которую они ввели последней в командной оболочке. Однако многие командные оболочки (включая bash и tcsh) имеют функции *управления заданиями* (job control), позволяющие запускать одновременно несколько команд или *заданий* (jobs) и, по мере надобности, переключаться между ними.

Управление заданиями может быть полезно, если, например, вы редактируете большой текстовый файл и хотите временно прервать редактирование, чтобы сделать какую-нибудь другую операцию. С помощью функций управления заданиями можно временно покинуть редактор, вернуться к приглашению командной оболочки и выполнить какие-либо другие действия. Когда они будут сделаны, можно вернуться обратно к работе с редактором и обнаружить его в том же состоянии, в котором он был покинут. У функций управления заданиями есть ещё много полезных применений.

Передний план и фоновый режим

Задания могут быть либо на *переднем плане* (foreground), либо *фоновыми* (background). На переднем плане в любой момент времени может быть только одно задание. Задание на переднем плане — это то задание, с которым вы взаимодействуете; оно получает ввод с клавиатуры и посылает вывод на экран (если, разумеется, вы не перенаправили ввод или вывод куда-либо ещё). Напротив, фоновые задания не получают ввода с терминала; как правило, такие задания не нуждаются во взаимодействии с пользователем.

Некоторые задания исполняются очень долго, и во время их работы не происходит ничего интересного. Пример таких заданий — компилирование программ, а также сжатие больших файлов. Нет никаких причин смотреть на экран и ждать, когда эти задания выполнятся. Такие задания следует запускать в фоновом режиме. В это время вы можете работать с другими программами.

Для управления выполнением процессов в `Linux` предусмотрен механизм передачи *сигналов*. Сигнал — это способность процессов обмениваться стандартными короткими сообщениями непосредственно с помощью системы. Сообщение-сигнал не содержит никакой информации, кроме номера сигнала (для удобства вместо номера можно использовать предопределённое системой имя). Для того, чтобы передать сигнал, процессу достаточно задействовать системный вызов **kill()**, а для того, чтобы принять сигнал, не нужно ничего. Если процессу нужно как-то по-особенному реагировать на сигнал, он может зарегистрировать *обработчик*, а если обработчика нет, за него отреагирует система. Как правило, это приводит к немедленному завершению процесса, получившего сигнал. Обработчик сигнала запускается *асинхронно*, немедленно после получения сигнала, что бы процесс в это время ни делал.

Два сигнала — номер 9 (**KILL**) и 19 (**STOP**) — всегда обрабатывает система. Первый из них нужен для того, чтобы убить процесс наверняка (отсюда и название). Сигнал **STOP** *приостанавливает* процесс: в таком состоянии процесс не удаляется из таблицы процессов, но и не выполняется до тех пор, пока не получит сигнал 18 (**CONT**) — после чего продолжит работу. В командной оболочке Linux сигнал **STOP** можно передать активному процессу с помощью управляющей последовательности **Ctrl-Z**.

Сигнал номер 15 (**TERM**) служит для прерывания работы задания. При *прерывании* (interrupt) задания процесс погибает. Прерывание заданий обыч-

но осуществляется управляющей последовательностью **Ctrl-C**. Восстановить прерванное задание никаким образом невозможно. Следует также знать, что некоторые программы перехватывают сигнал **TERM** (при помощи обработчика), так что нажатие комбинации клавиш **Ctrl-C** (о) может не прервать процесс немедленно. Это сделано для того, чтобы программа могла уничтожить следы своей работы прежде, чем она будет завершена. На практике, некоторые программы вообще нельзя прервать таким способом.

Перевод в фоновый режим и уничтожение заданий

Рассмотрим команду yes, которая на первый взгляд может показаться бесполезной. Эта команда посылает бесконечный поток строк, состоящих из символа `y` на стандартный вывод. Посмотрим, как работает эта команда:

```
/home/larry# yes
y
y
y
y
y
```

Последовательность таких строк будет бесконечно продолжаться. Уничтожить этот процесс можно, отправив ему сигнал прерывания, т. е. нажав **Ctrl-C**. Чтобы на экран не выводилась эта бесконечная последовательность перенаправим стандартный вывод команды yes на `/dev/null`. Как вы, возможно, знаете, устройство `/dev/null` действует как «чёрная дыра»: все данные, посланные в это устройство, пропадают. С помощью этого устройства очень удобно избавляться от слишком обильного вывода некоторых программ.

```
/home/larry# yes > /dev/null
```

Теперь на экран ничего не выводится. Однако и приглашение командной оболочки также не возвращается. Это происходит потому, что команда yes все ещё работает и посылает свои сообщения, состоящие из букв `y` на `/dev/null`. Уничтожить это задание также можно, отправив ему сигнал прерывания.

Допустим теперь, что вы хотите, чтобы команда yes продолжала работать, но при этом и приглашение командной оболочки должно вернуться на экран, так чтобы вы могли работать с другими программами. Для этого можно команду yes перевести в фоновый режим, и она будет там работать, не общаясь с вами.

Один способ перевести процесс в фоновый режим — приписать символ `&` к концу команды. Пример:

```
/home/larry# yes > /dev/null &
[1]+ 164
/home/larry#
```

Сообщение `[1]` представляет собой *номер задания* (job number) для процесса yes. Командная оболочка присваивает номер задания каждому исполняемому заданию. Поскольку yes является единственным исполняемым заданием, ему присваивается номер 1. Число `164` является идентификационным номером, соответствующим данному процессу (PID), и этот номер также

дан процессу системой. Как мы увидим дальше, к процессу можно обращаться, указывая оба этих номера.

Итак, теперь у нас есть процесс команды yes, работающий в фоне, и непрерывно посылающий поток из букв у на устройство `/dev/null`. Для того, чтобы узнать статус этого процесса, нужно исполнить команду **jobs**, которая является внутренней командой оболочки.

```
/home/larry# jobs
[1]+  Running                 yes >/dev/null  &
/home/larry#
```

Для того, чтобы узнать статус задания, можно также воспользоваться командой **ps**, как это было показано выше.

Для того, чтобы передать процессу сигнал (чаще всего возникает потребность *прервать* работу задания) используется утилита **kill**. В качестве аргумента этой команде даётся либо номер задания, либо PID. Необязательный параметр — номер сигнала, который нужно отправить процессу. По умолчанию отправляется сигнал **TERM**. В рассмотренном выше случае номер задания был 1, так что команда **kill %1** прервёт работу задания. Когда к заданию обращаются по его номеру (а не PID), тогда перед этим номером в командной строке нужно поставить символ процента («%»).

Теперь введём команду **jobs** снова, чтобы проверить результат предыдущего действия:

```
/home/larry# jobs 1
[1]+     Stopped          yes > /dev/null &
```

Фактически задание уничтожено, и при вводе команды jobs следующий раз на экране о нем не будет никакой информации.

Уничтожить задание можно также, используя идентификационный номер процесса (PID). Этот номер, наряду с идентификационным номером задания, указывается во время старта задания. В нашем примере значение PID было 164, так что команда **kill 164** была бы эквивалентна команде **kill %1**. При использовании PID в качестве аргумента команды kill вводить символ «%» не требуется.

Приостановка и продолжение работы заданий

Запустим сначала процесс командой yes на переднем плане, как это делалось раньше:

```
/home/larry# yes > /dev/null
```

Как и ранее, поскольку процесс работает на переднем плане, приглашение командной оболочки на экран не возвращается.

Теперь вместо того, чтобы прервать задание комбинацией клавиш **Ctrl-C**, задание можно *приостановить* (suspend, буквально — `подвесить`), отправив ему сигнал **STOP**. Для приостановки задания надо нажать соответствующую комбинацию клавиш, обычно это **Ctrl-Z**.

```
/home/larry# yes > /dev/null
Ctrl-Z[1]+  Stopped yes      >/dev/null
/home/larry#
```

Приостановленный процесс попросту не выполняется. На него не тратятся вычислительные ресурсы процессора. Приостановленное задание можно запустить выполняться с той же точки, как будто бы оно и не было приостановлено.

Для возобновления выполнения задания на переднем плане можно использовать команду **fg** (от слова `foreground` — передний план).

```
/home/larry# fg
yes >/dev/null
```

Командная оболочка ещё раз выведет на экран название команды, так что пользователь будет знать, какое именно задание он в данный момент запустил на переднем плане. Приостановим это задание ещё раз нажатием клавиш **Ctrl-Z**, но в этот раз запустим его в фоновый режим командой **bg** (от слова background — фон). Это приведёт к тому, что данный процесс будет работать так, как если бы при его запуске использовалась команда с символом `&` в конце (как это делалось в предыдущем разделе):

```
/home/larry# bg
[1]+ yes $>$/dev/null  &
/home/larry#
```

При этом приглашение командной оболочки возвращается. Сейчас команда **jobs** должна показывать, что процесс **yes** действительно в данный момент работает; этот процесс можно уничтожить командой **kill**, как это делалось раньше.

Для того, чтобы приостановить задание, работающее в фоновом режиме, нельзя воспользоваться комбинацией клавиш **Ctrl-Z**. Прежде, чем приостанавливать задание, его нужно перевести на передний план командой **fg** и лишь потом приостановить. Таким образом, команду **fg** можно применять либо к приостановленным заданиям, либо к заданию, работающему в фоновом режиме.

Между заданиями в фоновом режиме и приостановленными заданиями есть большая разница. Приостановленное задание не работает — на него не тратятся вычислительные мощности процессора. Это задание не выполняет никаких действий. Приостановленное задание занимает некоторый объем оперативной памяти компьютера, через некоторое время ядро откачает эту часть памяти на жёсткий диск «до востребования». Напротив, задание в фоновом режиме выполняется, использует память и совершает некоторые действия, которые, возможно, вам требуются, но вы в это время можете работать с другими программами.

Задания, работающие в фоновом режиме, могут пытаться выводить некоторый текст на экран. Это будет мешать работать над другими задачами.

```
/home/larry# yes &
```

Здесь стандартный вывод не был перенаправлен на устройство `/dev/null`, поэтому на экран будет выводится бесконечный поток символов `y`. Этот поток невозможно будет остановить, поскольку комбинация клавиш **Ctrl-C** не воздействует на задания в фоновом режиме. Для того чтобы оста-

новить эту выдачу, надо использовать команду **fg**, которая переведёт задание на передний план, а затем уничтожить задание комбинацией клавиш **Ctrl-C**.

Сделаем ещё одно замечание. Обычно командой **fg** и командой **bg** воздействуют на те задания, которые были приостановлены последними (эти задания будут помечены символом + рядом с номером задания, если ввести команду **jobs**). Если в одно и то же время работает одно или несколько заданий, задания можно помещать на передний план или в фоновый режим, задавая в качестве аргументов команды **fg** или команды **bg** их идентификационный номер (job ID). Например, команда **fg %2** помещает задание номер 2 на передний план, а команда **bg %3** помещает задание номер 3 в фоновый режим. Использовать PID в качестве аргументов команд **fg** и **bg** нельзя.

Более того, для перевода задания на передний план можно просто указать его номер. Так, команда **%2** будет эквивалентна команде **fg %2**.

Важно помнить, что функция управления заданием принадлежит оболочке. Команды **fg**, **bg** и **jobs** являются внутренними командами оболочки. Если, по некоторой причине, вы используете командную оболочку, которая не поддерживает функции управления заданиями, то вы в ней этих (и подобных) команд не отыщете.

5.2 Задание для практического занятия

2.1. Изучить команды управления процессами в Linux.
2.2. Создать обработчик сигнала в командной оболочке bash.

5.3 Задание для самостоятельной работы

1. Все действия выполняйте исключительно с помощью клавиатуры.
2. Выведите список всех выполняющихся процессов.
3. Оставить в выводе предыдущей команды только процессы пользователя **root.**
4. Выведите и отсортируйте список всех процессов по убыванию объема используемой памяти.
5. Выведите информацию о процессах одной команды (например, bash).
6. Вывести только PID, терминал и команду процессов в системе.
7. Запустите пять заданий в фоновом режиме.
8. Просмотрите состояние заданий.
9. Послать задаче сигнал STOP и возобновить выполнение сигналом CONT.
10. Послать разным задачам сигналы TERM, INT, QUIT, KILL и вывести состояние задач сразу после сигнала.
11. Запустите три задания в фоновом режиме.

12. Выведите все три задания из фонового режима командой **fg** (чтобы вернуться к приглашению командной строки остановите задачу на переднем плане).
13. Продолжите выполнение одного из заданий, выведенных из фонового режима.
14. Верните задание из предыдущего пункта в фоновый режим.
15. Переведите оставшиеся два задания в фоновый режим и отправьте сигналы продолжения выполнения.
16. Запустите в фоне две задачи: текстовый редактор **nano** и файловый менеджер **mc**.
17. Создайте файл file.txt с помощью редактора **nano**, переведите его в фоновый режим и просмотрите содержимое файла с помощью встроенного просмотрщика **mc**.
18. Переведите **mc** в фоновый режим. Выведите на экран список фоновых процессов и запишите его в журнал.
19. Удалите фоновый процесс **nano** и запишите в журнал список фоновых процессов.
20. Выведите на экран подробный список процессов текущего терминала и запишите его в журнал.
21. Отправьте сигнал SIGKILL процессу **mc** и запишите новый список процессов в журнал.
22. Выведите список процессов с помощью утилиты **top**, запустите любую программу с помощью меню «Приложения», проследите как изменится список процессов, выводимый утилитой **top**, а также загрузка процессора.
23. Запишите список всех доступных сигналов в журнал.
24. Создайте обработчик сигнала, который дописывает в файл ~/ps.log список процессов текущего терминала при нажатии клавиш Ctrl+\ и удаляет файл ~/ps.log при нажатии Ctrl+Z.
25. Покажите результаты работы преподавателю. Удалите файл журнала.

5.4 Контрольные вопросы

1. Дайте опеределение *процессу*
2. Что нужно сделать для приостановления задания, работающее в фоновом режиме?
3. Какие команды управления процессами в Linux?
4. Что делает команда yes?
5. Задания в фоновом режиме и приостановленные задания, в чем их отличие?
6. Для чего служит системный вызов kill()?

6 ЛАБОРАТОРНАЯ РАБОТА № 6. ПЛАНИРОВАНИЕ ЗАДАНИЙ И УПРАВЛЕНИЕ ПРИОРИТЕТАМИ ПРОЦЕССОВ

Цель работы: изучение команд и способов управления процессами в Linux.

6.1 Теория

6.1.1 Отложенное выполнение заданий

Для однократного запуска процесса в нужный момент времени используется команда **at**. Формат запуска команды: **at** <время>, где
<время> — now + <n> minutes, hours, days, weeks; month, day, HH:MM, midnight, noon, teatime (16:00), am, pm, today, tomorrow.
Примеры:

```
$ at now + 2 minutes
$ at 10:00
$ at 10:00 tomorrow
```

После запуска данной команды появляется приглашение at>, где нужно ввести отложенную команду (или несколько команд — каждую с новой строки). Завершается ввод отложенных команд нажатием клавиш Ctrl+D.
Основные команды для работы с отложенными заданиями:

- **atq** — вывести список отложенных заданий;
- **atrm** <номер задания> — удалить задание с указанным номером;
- **at** <время> -f <имя файла> — перечень команд, запускаемых при выполнении отложенного задания, указывается в файле.
- **batch** <команда> — выполнить команду, когда средняя загрузка системы уменьшится до 0.8 (см. файл /proc/loadavg).

Вся выводимая отложенным процессом информация записывается в письмо и отправляется пользователю с помощью встроенной системы почты. Поэтому для вывода данных на экран для каждой необходимо указать перенаправление в файл текущего терминала, например, **date > /dev/tty1**.
Список отложенных заданий сохраняется в каталоге /var/spool/cron/atjobs. Списки пользователей, которым разрешено и запрещено использовать команду at, хранятся в файлах /etc/at.allow и /etc/at.deny. Если файл /etc/at.allow существует, то право на использование команды at имеют только пользователи, указанные в этом файле. В противном случае, право на использование данной командой имеют пользователи, не указанные в файле /etc/at.deny.

6.1.2. Регулярное выполнение заданий

Регулярный запуск процессов с заданным интервалом осуществляет планировщик **cron**. Планировщик **cron** является демоном, т.е. непосредственно с

пользователем не взаимодействует. Для управления работой планировщика используется команда **crontab**:

• **crontab** <имя файла> — установка таблицы заданий;
• **crontab –l** — просмотр таблицы заданий;
• **crontab –r** — удаление таблицы заданий;
• **crontab** <файл или опция> **–u** <пользователь> — выполнение действий с таблицей заданий указанного пользователя.

Таблица заданий должна содержать 6 колонок, разделенных пробелами:
1. минута часа (0–59);
2. час суток (0–23);
3. день месяца (1–31);
4. месяц (1–12);
5. день недели (0–6, начиная с воскресенья);
6. исполняемая команда.

Первые пять колонок определяют регулярность запуска процесса, шестая колонка — команду. Если процесс необходимо запускать каждый месяц, день, час или минуту, в соответствующей колонке указывается символ *. Например, для ежедневного запуска команды ps в 12:00 необходимо указать следующую строку: 0 12 * * * ps.

Таблицы заданий сохраняются в каталоге /var/spool/cron/crontabs. Списки пользователей, которым разрешено и запрещено использовать cron, хранятся в файлах /etc/cron.allow и /etc/cron.deny. Если файл /etc/cron.allow существует, то право на использование cron имеют только пользователи, указанные в этом файле. В противном случае, право на использование cron имеют пользователи, не указанные в файле /etc/cron.deny.

6.1.3 Управление приоритетами процессов

Управление приоритетами процессов в Linux осуществляется с помощью команд nice и renice. Данные команды задают относительный приоритет процесса в диапазоне от –20 (наивысший приоритет) до +19 (наинизший). Приоритет по умолчанию — 0. Отрицательный приоритет может устанавливать только суперпользователь (root).

• nice –n <приоритет> <команда> — запуск команды с указанным приоритетом;
• renice <приоритет> <PID> — установка процессу с идентификатором PID указанного приоритета;
• renice <приоритет> -u <пользователь> — установка всем процессам пользователя указанного приоритета.

6.2 Задание для практического занятия

1. Изучить команды для работы с планировщиком заданий в Linux.
2. Изучить влияние приоритета на производительность процесса.

6.3 Задание для самостоятельной работы

1. Все действия выполняйте исключительно с помощью клавиатуры.
2. Определите имя файла текущего терминала командой tty (при работе в текстовом режиме имя файла основного терминала /dev/tty1).
3. Создайте отложенное на 2 минуты задание, выводящее подробный список всех процессов пользователя student на экран.
4. Дождитесь вывода списка процессов на экран.
5. Установите отложенные задания, выводящие сообщения: «конец пары», «обед», «пора спать!!!» (с одновременным выключением компьютера командой halt — только для суперпользователя root).
6. Удалите задание, выводящее на экран сообщение «обед».
7. Создайте задание, выводящее на экран текущее время в формате ЧЧ:ММ каждую минуту.
8. Напишите и скомпилируйте программу output, которая в бесконечном цикле выводит символ «a» в файл, имя которого указано в первом аргументе командной строки.
9. Запустите одновременно два фоновых процесса и через некоторое время принудительно завершите их.

Запуск:

```
./output a.txt & ./output b.txt &
```

Завершение:

```
kill PID1 & kill PID2 &
```

где PID1 и PID2 — идентификаторы двух запущенных процессов.

10. Сравните размер файлов a.txt и b.txt.
11. Запустите одновременно два фоновых процесса с разными приоритетами (0 и +10 соответственно) и через некоторое время принудительно завершите их.
12. Сравните размер файлов a.txt и b.txt.
13. Покажите результаты работы преподавателю. Удалите все созданные файлы и запланированные задания.__

6.4 Контрольные вопросы

1. Что осуществляет планировщик cron?
2. Какие основные команды для работы с планировщиком заданий в Linux
3. Что происходит с выводимой отложенным процессом информацией?
4. Для чего нужна команда at? Какой у ней формат?
5. С помощью каких команд осуществляется управление приоритетами процессов в Linux?
6. Каково влияние приоритета на производительность процесса?

7 ЛАБОРАТОРНАЯ РАБОТА № 7. НАСТРОЙКА СЕТИ

Цели работы: научиться получать информацию о сетевых взаимодействиях, научиться использовать удаленный доступ.

7.1 Теория

Взаимодействие с сетью в Linux осуществляется через сетевые интерфейсы. Любые данные, которые компьютер отправляет в сеть или получает из сети проходят через сетевой интерфейс. Сетевой интерфейс – физическое или виртуальное устройство, предназначенное для передачи данных между программами через компьютерную сеть.

Примеры сетевых интерфейсов:

- Физические интерфейсы сетевых карт и телекоммуникационных устройств (коммутаторов, маршрутизаторов и так далее)
- Петлевые интерфейсы для обмена данными между процессами на одном компьютере или управляемом сетевом устройстве. Для них выделена специальная подсеть 127.0.0.0/8
- Туннели — для инкапсуляции протокола того же или более низкого уровня в другой протокол
- Интерфейсы виртуальных сетей (VLAN)

Каждый интерфейс в сети может быть однозначно идентифицирован по его адресу. Разные сетевые протоколы используют разные системы адресации, например, MAC-адреса в Ethernet или IP-адреса в IP.

Настройка сетевых интерфейсов в UNIX-подобных системах традиционно выполняется с помощью команды **ifconfig**. В Linux рекомендуется использовать альтернативу – команду **ip**, т.к. разработка пакета, содержащего **ifconfig** прекращена.

Рассмотрим команды **ifconfig** и **ip**.

ifconfig – устаревшая команда для конфигурации сетевых интерфейсов из пакета net-tools (пакет больше не разрабатывается с 2001 года). Любое действие, выполняемое **ifconfig**, может быть выполнено командой **ip**.

ip – команда из пакета **iproute2**. По функциональным возможностям **ip** превосходит **ifconfig**. **ip** предназначена для просмотра и изменения маршрутизации, настроек устройств и их интерфейсов, политик маршрутизации и тоннелей.

`ifconfig`	`ip addr, ip link`	Просмотр текущего состояния сетевых итерфейсов
`ifconfig [interface stats]`	`ip -s link`	Просмотр статистики подключений

`ifconfig [interface alias] [ip] netmask [mask] up`
или
`ifconfig [interface alias] up`
	`ip link set [ip]/[mask-digits] dev [interface alias] up`
	или
	`ip link set [interface alias] up`
		Включение сетевого интерфейса, для выключения используется ключевое слово down вместо up

ip link – обращается к информации об интерфейсах без привязки к протоколам.

ip addr – информация об интерфейсах с привязкой к протоколу.

Например, чтобы изменить параметры сетевого интерфейса, работающего по протоколу IPv4, можно использовать следующие команды:

- изменение IP-адреса

```
ifconfig [interface alias] [new ip]
```

- изменение маски подсети

```
ifconfig [interface alias] netmask [new netmask]
```

- изменение шлюза

```
ifconfig [interface alias] gateway [new gateway]
```

Согласно модели OSI, сетевые интерфейсы представляют собой канальный и сетевой уровень. Сетевой и транспортный уровень связываются механизмом сокетов. **Socket** – абстрактное программное понятие, используемое для обозначения в прикладной программе конечной точки канала связи с коммуникационной средой, образованной вычислительной сетью. При использовании протоколов TCP/IP или UDP/IP можно говорить, что сокет является средством подключения прикладной программы к порту локального узла сети, где сокет однозначно определяется своим сетевым протоколом (представлен номером из файла /etc/protocols/) и номером порта. Сокет-интерфейс представляет собой набор системных вызовов и/или библиотечных функций языка программирования С, разделенных на четыре группы:

1. локального управления;
2. установления связи;
3. обмена данными (ввода/вывода);
4. закрытия связи.

Для просмотра состояния сокетов из командной строки можно использовать команды **netstat** и **ss**.

Просмотр всех соединений по всем протоколам

```
netstat –a
```

Просмотр только прослушиваемых сокетов

```
netstat –l
```

Просмотр статистики по протоколам

```
netstat –s
```

Для детального рассмотрения всего, что передается по сети, или для учебно-исследовательских целей, чтобы узнать как взаимодействуют между собой объекты сети, рассмотрим команду **tcpdump**.

tcpdump выводит заголовки пакетов проходящих через сетевой интерфейс, которые совпадают с булевым выражением. Он может также быть запущен с ключем -w, который указывает сохранять данные пактов в файл для дальнейшего исследования, и/или с ключем -r, который указывает читать сохраненные пакеты из файла, вместо чтения пакетов из сетевого интерфейса.

tcpdump будет, если не запущен с ключем -c, продолжать собирать пакеты до тех пор, пока не будет прерван сигналом SIGINT или сигналом SIGTERM. Если запуск был с ключем -c, то сбор пакетов будет продолжаться до тех пор, пока не произойдет прерывание сигналом SIGINT или SIGTERM или пока не будет обработано определенное количество пакетов.

Когда **tcpdump** закончит сбор пакетов, то будет сообщено о количестве:

- пакетов "полученных фильтром" (received by filter) (обычно, это пакеты, соответствующие выращению и обработанные командой **tcpdump**);
- пакетов "отброшенных ядром" (dropped by kernel) (это число пакетов, которые были отброшены, в зависимости от механизма сбора пакетов (недостаточного объема буферов) на той ОС, где запускается tcpdump, ОС предоставит эту информацию приложению или нет, и тогда будет выведено число 0)

Рассмотрим фильтрующее выражение. По нему выбираются пакеты из общего потока. Если оно не указано, то будут выбираться и выводится все пакеты, идущие через интерфейс. Иначе, будут обработаны только те пакеты, для которых проверка с выражением выдаст значение «истина».

Выражение состоит из одного или более примитивов. Примитивы обычно состоят из ID (имя или номер) следующего за одним или более классификаторами. Различают три вида классификаторов:

- **type** – вид ID. Возможные значения host, net или port. Пример: 'host foo', 'net 128.3', 'port 20'. Если классификатор **type** не указан, то подразумевается host.
- **dir** – определяет конкретное направление передачи "к" и/или "от" ID. Возможны значения src, dst, src or dst, and src and dst. Пример, 'src foo', 'dst net 128.3', 'src or dst port ftp-data'. Если не указан, то подразумевается src or dst. Для соединений нулевого ('*null*') уровня (к примеру, протокол точка-точка, такой как slip) указанием направления могут быть классификаторы inbound и outbound.
- **proto** – ограничивает совпадение конкретным протоколом. Возможные протоколы: ether, fddi, tr, ip, ip6, arp, rarp, decnet, tcp и udp. Пример, 'ether src foo', 'arp net 128.3', 'tcp port 21'. Если классификатор **proto** не указан, то подразумеваются все перечисленные типы протоколов. Например, 'src foo означает '(ip or arp or rarp) src foo', 'net bar' означает '(ip or arp or rarp) net bar', а 'port 53' означает '(tcp or udp) port 53'.

Также существует несколько специальных примитивов – ключевых слов: gateway, broadcast, less, greater и арифметические выражения.

Более сложные фильтрующие выражения могут быть построены с помощью слов and, or и not, объединяющих примитивы. Пример, 'host foo and not port ftp and not port ftp-data'. Чтобы уменьшить количество вводимой информации, идентичные списки классификаторов могут быть опущены. Пример, 'tcp dst port ftp or ftp-data or domain' это тоже самое, что и 'tcp dst port ftp or tcp dst port ftp-data or tcp dst port domain'. Типичные результаты работы tcpdump -ttt –c 10:

```
student@debian:~$ sudo tcpdump -ttt -c 10
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
00:00:00.000000 IP debian.56006 > 192.168.0.1.domain: 55528+ A? duckduckgo.com.
 (32)
00:00:00.000265 IP debian.56006 > 192.168.0.1.domain: 38911+ AAAA? duckduckgo.c
om. (32)
00:00:00.027411 IP 192.168.0.1.domain > debian.56006: 55528 6/0/0 A 54.229.105.
203, A 46.51.197.89, A 54.229.115.42, A 176.34.155.20, A 176.34.135.167, A 176.
34.131.233 (128)
00:00:00.000687 IP 192.168.0.1.domain > debian.56006: 38911 0/1/0 (88)
00:00:00.000714 IP debian.34195 > ec2-54-229-105-203.eu-west-1.compute.amazonaw
s.com.https: Flags [S], seq 2510782406, win 29200, options [mss 1460,sackOK,TS
val 4935525 ecr 0,nop,wscale 7], length 0
00:00:00.080885 IP ec2-54-229-105-203.eu-west-1.compute.amazonaws.com.https > d
ebian.34195: Flags [S.], seq 1280001, ack 2510782407, win 65535, options [mss 1
460], length 0
00:00:00.000123 IP debian.34195 > ec2-54-229-105-203.eu-west-1.compute.amazonaw
s.com.https: Flags [.], ack 1, win 29200, length 0
00:00:00.000507 IP debian.34195 > ec2-54-229-105-203.eu-west-1.compute.amazonaw
s.com.https: Flags [P.], seq 1:186, ack 1, win 29200, length 185
00:00:00.000301 IP ec2-54-229-105-203.eu-west-1.compute.amazonaws.com.https > d
ebian.34195: Flags [.], ack 186, win 65535, length 0
00:00:00.046040 IP debian.38477 > 192.168.0.1.domain: 42295+ PTR? 1.0.168.192.i
n-addr.arpa. (42)
10 packets captured
231 packets received by filter
0 packets dropped by kernel
```

Первое поле – поле времени, т. к. запуск осуществлялся с ключом "-ttt", то это разница в микросекундах между этим пакетом и предыдущим. Потом идет протокол (здесь – IP), IP-адрес или имя (здесь имя debian) отправителя пакета, через точку может указываться порт (здесь в первом пакете – 56006). После знака ">", указывается получатель пакета (или его имя) и также порт. Затем будет идти служебная информация идущая в пакете. В служебной информации может быть указано либо состояние флагов в пакете, либо расшифрованная информация.

Сервер защищенных соединений (ssh)

Протокол *SSH (Secure SHell)* позволяет получить удаленный доступ к компьютеру по безопасному соединению. *SSH* реализуется двумя приложениями: сервером и клиентом. При подключении клиент проходит процедуру аутентификации и между клиентом и сервером устанавливается защищенное соединение.

В Linux для реализации *SSH* используется программное обеспечение *OpenSSH*. Для установки сервера в *Debian* можно использовать команду:

```
# apt-get install openssh-server
```

В процессе установки сервер автоматически прописывается в автозагрузку и запускается. По умолчанию сервер слушает на TCP-порту с номером 22 на всех интерфейсах. Можно убедиться в этом командой **ss** или **netstat**.

Для остановки/запуска/перезапуска сервера можно использовать команду service с параметром stop/start/restart соответственно, например для перезапуска:

```
# service ssh restart
```

Конфигурационный файл сервера **sshd_config** расположен в каталоге **/etc/ssh**. В настройках по умолчанию указано использовать аутентификацию на основе публичных ключей (PubkeyAuthentication yes) или логина и пароля (PasswordAuthentication yes), разрешать удаленное подключение суперпользователя (PermitRootLogin yes), использовать *SSH-2* (Protocol 2), не разрешать пустые пароли (PermitEmptyPasswords no) и т. д. Для применения внесенных изменений в конфигурацию необходимо перезапустить сервер.

Для того, чтобы сервер ssh слушал, например, на адресе 192.168.56.102, необходимо в файле */etc/ssh/sshd_config* раскомментировать и исправить опцию ListenAddress: ListenAddress 192.168.56.102

Затем перезапустить сервер ssh. Результат можно проверить командой ss.

Для получения удаленного доступа к командному интерпретатору на клиентской системе необходимо использовать утилиту ssh, которая обычно присутствует в базовой установке системы (ее также можно установить, используя команду apt-get install openssh-client). В качестве параметра надо указать имя пользователя и IP-адрес (имя удаленной системы) используя символ @ в качестве разделителя:

```
$ ssh root@192.168.56.102
The authenticity of host '192.168.56.102 (192.168.56.102)' can't be established.
RSA key fingerprint is d7:33:4d:e3:0b:e2:ec:57:48:34:f1:77:88:ab:77:de.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.56.102' (RSA) to the list of known hosts.
root@192.168.56.102's password:
Linux ssl1 2.6.32-5-686 #1 SMP Wed Jan 12 04:01:41 UTC 2011 i686
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Apr  6 13:19:19 2011 from 192.168.56.1
root@ssl1:~#
```

Листинг 8.2. Первое подключение к удаленной системе

При первом подключение будет выдано предупреждение о неизвестном узле. При ответе yes, удаленный узел будет добавлен в список известных узлов (файл **~/.ssh/known_hosts**) и при последующих подключениях предупреждение не будет выдаваться. Затем нужно ввести пароль. После успешного ввода пароля будет предоставлен доступ к интерфейсу командной строки удаленной системы. Для завершения удаленного сеанса следует использовать команду exit.

Для копирования файлов по протоколу *ssh* можно использовать утилиту **scp**, при этом на одной из систем должен быть установлен *ssh-сервер*.

```
# scp root@192.168.56.102:/home/student/test.txt .
root@192.168.56.102's password:
test.txt                  100% 318KB   8.1MB/s   00:01
```

В данном примере файл копируется из указанного пути *(*/home/student/test.txt*)* удаленной системы (192.168.56.102) в текущий каталог локальной системы (.). Для этого используется учетная запись суперпользователя удаленной системы.

```
$ scp known_hosts root@192.168.56.102:/root
```

В данном примере файл (*known_hosts*) копируется из текущего каталога локальной системы в каталог суперпользователя *(/root*) удаленной системы.
Для копирования каталогов в команде scp нужно использовать ключ -r.

7.2 Задание для практического занятия

1. Подключитесь к соседнему компьютеру (клиенту) через ssh.
2. Скопируйте с хоста-клиента на хост-сервер отдельно файл и каталог с файлами и вложенными каталогами.

7.3 Задание для самостоятельной работы

3. Все действия выполняйте исключительно с помощью клавиатуры.
4. Определите, через какой интерфейс происходит доступ в интернет и запишите его параметры в файл.
5. Отключите интерфейс из предыдущего пункта и проверьте наличие доступа в интернет.
6. Восстановите интерфейс для доступа в интернет.
7. Поменяйте IP-адрес интерфейса и проверьте его доступность по новому адресу командой ping с другого компьютера.
8. Проверьте работу ssh сервера. Установите его, если он не установлен.
9. Командой netstat или ss убедитесь, что ssh прослушивает порт на внешнем интерфейсе.
10. Подключитесь к соседнему компьютеру (клиенту) через ssh.
11. Скопируйте с хоста-клиента на хост-сервер отдельно файл и каталог с файлами и вложенными каталогами.
12. Скопируйте с хоста-сервера на хост-клиент отдельно файл и каталог с файлами и вложенными каталогами.
13. Используйте команду tcpdump для ssh-сессии и запишите результат в файл. Проанализируйте пакеты.

7.4 Контрольные вопросы

1. Для считывания аргументов командной строки используются?
2. Как скомпилировать проект?
3. Какие функции стандартной библиотеки для работы с файловой системой?
4. Как выводить список файлов в два столбца: имя файла, размер файла?

8 ЛАБОРАТОРНАЯ РАБОТА № 8. ОСНОВЫ ПРОГРАММИРОВАНИЯ В LINUX

Цели работы:

1. Получить практику работы с компилятором gcc в операционной системе Linux.

2. Научиться использовать функции стандартной библиотеки для работы с файловой системой.

8.1 Теория

Для считывания аргументов командной строки используйте следующее объявление функции main:

```
void main(int argc, char* argv[])
```

В переменной argc содержится количество аргументов командной строки (один аргумент существует всегда – имя исполняемого файла программы). В массиве строк argv[] содержатся аргументы командной строки, например, argv[0] – имя исполняемого файла программы, argv[1] – первый аргумент и т.д.

Пример. Написать программу вывода списка файлов из текущего каталога с указанием их размера. Формат вызова программы: list <путь>.

```
#include <stdio.h> #include <stdlib.h> #include <dirent.h>
#include <sys/stat.h>

void main(int argc, char* argv[])
{
     DIR* d;
     struct dirent * entry; struct stat st;

     // проверка количества аргументов if (argc != 2)
     {
          printf("Формат вызова: list путь\n"); return;
     }

     // открытие каталога d = opendir(argv[1]);
     if (d == NULL) // если каталог не найден
     {
          printf("Каталог не найден\n"); return;
     }

     // чтение каталога
     while((entry = readdir(d)) != NULL)
     {
```

```
            // получение информации о файле stat(entry-
        >d_name, &st);
            // проверка, обычный ли это файл if
        (S_ISREG(st.st_mode))
            {
                // вывод на экран имени и размера файла
             printf("%20s\t%d", entry->d_name, st.st_size);
            }
        }

        closedir(d); // закрытие каталога
    }
```

8.2 Задание для практического занятия

1. Разработать алгоритм и программу согласно варианту задания с использованием языка Си. Программа должна принимать аргументы командной строки.

2. Проверить функционирование разработанной программы на наборе тестов.

3. Провести анализ этапов выполнения задачи.

4. Подготовить и оформить отчет по выполнению лабораторной работы. Отчет должен содержать титульный лист, техническое задание, блок- схемы алгоритмов, исходный текст программы с комментариями, тесты, результаты их выполнения и выводы по работе.

Порядок выполнения работы

Создайте файл исходного текста программы с помощью редактора nano.

1. Разработайте алгоритм программы в соответствии с вариантом задания.
2. Разбейте алгоритм на несколько логических блоков.
3. Создайте исходный текст программы, реализующей разработанный алгоритм.
4. Скомпилируйте проект командой gcc program.c – program, гле program — название программы.
5. Устраните синтаксические ошибки, если такие имеются.
6. Разработайте набор тестов с учетом особенностей реализации алгоритма.
7. Проверьте правильность функционирования программы с использованием набора тестов.
8. Устраните семантические ошибки в программе, если такие имеются.

8.3 Задание для самостоятельной работы

1. 1 Написать программу поиска всех файлов в указанном каталоге, содержащих заданное слово. Формат вызова программы: wordsearch

<путь> <искомое слово>. Программа должна выводить имена всех файлов в один столбец.

2. Написать программу поиска файла по имени в указанном каталоге и во всех вложенных. Формат вызова программы: filesearch <путь> <имя файла>. Программа должна вывести путь к искомому файлу.

3. Написать программу вывода списка имен всех файлов из указанного каталога и всех вложенных в алфавитном порядке с указанием размера файла. Формат вызова программы: filelist <путь>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.

4. Написать программу вывода списка имен всех файлов из указанного каталога и всех вложенных в порядке возрастания или убывания размера файла. Формат вызова программы: sizelist <путь> <-a по возрастанию или –d по убыванию>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.

5. Написать программу вывода иерархического списка всех вложенных каталогов, начиная с указанного. Формат вызова программы: catlist <путь>. Программа должна выводить иерархический список каталогов по строкам: в первой строке — корневой каталог, во второй — его подкаталоги, далее — их подкаталоги и т.д.

6. Написать программу вывода списка имен всех файлов из указанного каталога в алфавитном порядке с группировкой по расширению (список сортируется по возрастанию расширения, внутри каждой группы с одинаковым расширением — в порядке возрастания имени файла) с указанием размера файла. Формат вызова программы: alphalist <путь>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.

7. Написать программу вывода списка имен всех файлов из указанного каталога в алфавитном порядке (по расширению), внутри каждой группы файлов с одинаковым расширением — в порядке возрастания размера файла с указанием размера файла. Формат вызова программы: Xlist <путь>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.

8. Написать программу форматирования файла, содержащего дробные числа. Входные данные — файл с дробными числами в произвольном формате, выходные данные — файл с дробными числами в формате XXX.XX (например, 001.00) в один столбец. Формат вызова программы: floatformat –s <исходный файл> –d <выходной файл>. Если входной или выходной файл не указаны, то данные вводятся с клавиатуры или выводятся на экран.

9. Написать программу автоматического создания и удаления каталогов по заданному шаблону. Шаблон имеет следующий вид: «каталог %d-й», где символ %d должен быть заменен на число. Формат вызова программы: autodir <–c или –d> <шаблон> <начальный номер> <конечный номер>, для создания каталогов с именами cat1, cat2, cat3: autodir –c cat%d 1 3.

10. Написать программу автоматического создания и удаления файлов по заданному шаблону. В каждый файл должно быть записано его имя. Шаблон имеет следующий вид: «file%d.txt», где символ %d должен быть заменен на число. Формат вызова программы: autofile <–c (для создания) или –d (для удаления)> <шаблон> <начальный номер> <конечный номер>, для создания каталогов с именами file1.txt, file2.txt, file3.txt: autodir –c file%d.txt 1 3.
11. Написать программу поиска всех файлов в указанном каталоге, содержащих заданное число. Формат вызова программы: numbersearch
<путь> <искомое число>. Программа должна выводить имена всех файлов в один столбец.
12. Написать программу вывода списка имен всех файлов из указанного каталога и всех вложенных в алфавитном порядке (сначала по расширению, затем — по имени) с указанием размера файла. Формат вызова программы: fileextlist <путь>. Программа должна выводить список файлов в три столбца: имя файла, расширение файла, размер файла.
13. Написать программу измерения размера всех подкаталогов в заданном каталоге. Размер подкаталога определяется рекурсивно как сумма размеров всех его файлов, включая файлы во вложенных подкаталогах. Формат вызова программы: dirsize <путь>. Программа должна выводить список каталогов в два столбца: имя каталога, размер каталога.
14. Написать программу переноса всех текстовых файлов из одного каталога в другой с заменой всех букв внутри файла на прописные. Формат вызова программы: mvlowcase <откуда> <куда>. Пример: mvlowcase ~/A1 ~/A2.
15. Написать программу подсчета количества слов для всех текстовых файлов *.txt заданного каталога. Формат вызова программы: wordcount
<путь>. Программа должна выводить список файлов в два столбца: имя файла, количество слов в файле.
16. Написать программу вывода иерархического списка всех вложенных каталогов и файлов, начиная с указанного. Формат вызова программы: catfilelist <путь>. Программа должна выводить иерархический список каталогов по строкам: в первой строке — корневой каталог, во второй — его подкаталоги и файлы, далее — их подкаталоги и файлы и т.д.
17. Написать программу автоматического создания символических ссылок на все файлы из указанного каталога и всех его вложенных подкаталогов. Все ссылки должны быть сохранены в одном каталоге. Формат вызова программы: autosymlink <каталог с файлами> <каталог с ссылками>.
18. Написать программу поиска всех файлов в указанном каталоге и во всех вложенных подкаталогах, содержащих в своем имени указанную строку.Формат вызова программы: filenamesearch <путь> <искомая строка>. Программа должна выводить имена всех файлов в один столбец.
19. Написать программу поиска слова наибольшей длины для всех текстовых файлов *.txt заданного каталога. Формат вызова программы: wordlen

<путь>. Программа должна выводить список файлов в три столбца: имя файла, найденное слово и его длина.
20. Написать программу автоматического создания символических ссылок на все файлы из указанного каталога, содержащие в названии заданную строку. Все ссылки должны быть сохранены в одном каталоге. Формат вызова программы: filtersymlink <каталог с файлами> <строка> <каталог с ссылками>.
21. Написать программу копирования всех текстовых файлов из одного каталога в другой с заменой всех букв внутри файла на строчные. Формат вызова программы: cplowcase <откуда> <куда>. Пример: cplowcase ~/A1
~/A2.
22. Написать программу поиска файла по имени в указанном каталоге и во всех вложенных. Формат вызова программы: filesearch <путь> <имя файла>. Программа должна вывести путь к искомому файлу.
23. Написать программу вывода списка имен всех файлов из указанного каталога и всех вложенных в алфавитном порядке с указанием размера файла. Формат вызова программы: filelist <путь>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.
24. Написать программу вывода списка имен всех файлов из указанного каталога и всех вложенных в порядке возрастания или убывания размера файла. Формат вызова программы: sizelist <путь> <-a по возрастанию или
–d по убыванию>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.
25. Написать программу вывода иерархического списка всех вложенных каталогов, начиная с указанного. Формат вызова программы: catlist
<путь>. Программа должна выводить иерархический список каталогов по строкам: в первой строке — корневой каталог, во второй — его подкаталоги, далее — их подкаталоги и т.д.
26. Написать программу вывода списка имен всех файлов из указанного каталога в алфавитном порядке с группировкой по расширению (список сортируется по возрастанию расширения, внутри каждой группы с одинаковым расширением — в порядке возрастания имени файла) с указанием размера файла. Формат вызова программы: alphalist <путь>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.
27. Написать программу вывода списка имен всех файлов из указанного каталога в алфавитном порядке (по расширению), внутри каждой группы файлов с одинаковым расширением — в порядке возрастания размера файла с указанием размера файла. Формат вызова программы: Xlist
<путь>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.Написать программу форматирования файла, содержащего дробные числа. Входные данные — файл с дробными числами в произвольном формате, выходные данные — файл с дробными числами в формате
XXX.XX (например, 001.00) в один столбец. Формат вызова программы: floatformat –s <исходный файл> –d <выходной файл>. Если входной или вы-

ходной файл не указаны, то данные вводятся с клавиатуры или выводятся на экран.
28. Написать программу автоматического создания и удаления каталогов по заданному шаблону. Шаблон имеет следующий вид: «каталог %d-й», где символ %d должен быть заменен на число. Формат вызова программы: autodir <–c или –d> <шаблон> <начальный номер> <конечный номер>, для создания каталогов с именами cat1, cat2, cat3: autodir –c cat%d 1 3.
29. Написать программу автоматического создания символических ссылок на все файлы из указанного каталога, содержащие в названии заданную строку. Все ссылки должны быть сохранены в одном каталоге. Формат вызова программы: filtersymlink <каталог с файлами> <строка> <каталог с ссылками>.
30. Написать программу автоматического создания символических ссылок на все файлы из указанного каталога, содержащие в названии заданную строку. Все ссылки должны быть сохранены в одном каталоге. Формат вызова программы: filtersymlink <каталог с файлами> <строка> <каталог с ссылками>
31. Написать программу автоматического создания и удаления файлов по заданному шаблону. В каждый файл должно быть записано его имя. Шаблон имеет следующий вид: «file%d.txt», где символ %d должен быть заменен на число. Формат вызова программы: autofile <–c (для создания) или –d (для удаления)> <шаблон> <начальный номер> <конечный номер>, для создания каталогов с именами file1.txt, file2.txt, file3.txt: autodir –c file%d.txt 1 3.
32. Написать программу поиска всех файлов в указанном каталоге, содержащих заданное число. Формат вызова программы: numbersearch
<путь> <искомое число>. Программа должна выводить имена всех файлов в один столбец.
33. Написать программу вывода списка имен всех файлов из указанного каталога и всех вложенных в порядке возрастания или убывания размера файла. Формат вызова программы: sizelist <путь> <-a по возрастанию или
–d по убыванию>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.
34. Написать программу поиска всех файлов в указанном каталоге, содержащих заданное слово. Формат вызова программы: wordsearch
<путь> <искомое слово>. Программа должна выводить имена всех файлов в один столбец.

8.4 Контрольные вопросы

1. Для считывания аргументов командной строки используются?
2. Как скомпилировать проект?
3. Какие функции стандартной библиотекидля работы с файловой системой?
4. Как выводить список файлов в два столбца: имя файла, размер файла?

9 ЛАБОРАТОРНАЯ РАБОТА № 9. ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ ПО СЕТИ

Цель работы: научиться разрабатывать сетевые приложения в Linux

9.1 Теория

Примеры сетевых интерфейсов:

- Физические интерфейсы сетевых карт и телекоммуникационных устройств (коммутаторов, маршрутизаторов и так далее)
- Петлевые интерфейсы для обмена данными между процессами на одном компьютере или управляемом сетевом устройстве. Для них выделена специальная подсеть 127.0.0.0/8
- Туннели — для инкапсуляции протокола того же или более низкого уровня в другой протокол
- Интерфейсы виртуальных сетей (VLAN)

Каждый интерфейс в сети может быть однозначно идентифицирован по его адресу. Разные сетевые протоколы используют разные системы адресации, например, MAC-адреса в Ethernet или IP-адреса в IP.

Настройка сетевых интерфейсов в UNIX-подобных системах традиционно выполняется с помощью команды **ifconfig**. В Linux рекомендуется использовать альтернативу – команду **ip**, т.к. разработка пакета, содержащего **ifconfig** прекращена.

Рассмотрим команды **ifconfig** и **ip**.

ifconfig – устаревшая команда для конфигурации сетевых интерфейсов из пакета net-tools (пакет больше не разрабатывается с 2001 года). Любое действие, выполняемое **ifconfig**, может быть выполнено командой **ip**.

ip – команда из пакета **iproute2**. По функциональным возможностям **ip** превосходит **ifconfig**. **ip** предназначена для просмотра и изменения маршрутизации, настроек устройств и их интерфейсов, политик маршрутизации и тоннелей.

`ifconfig`	*`ip addr, ip link`*	Просмотр текущего состояния сетевых итерфейсов
`ifconfig [interface stats]`	`ip -s link`	Просмотр статистики подключений

`ifconfig [interface alias] [ip] netmask [mask] up`
или
`ifconfig [interface alias] up`
	`ip link set [ip]/[mask-digits] dev [interface alias] up`
	или
	`ip link set [interface alias] up`
		Включение сетевого интерфейса, для выключения используется ключевое слово down вместо up

ip link – обращается к информации об интерфейсах без привязки к протоколам.

ip addr – информация об интерфейсах с привязкой к протоколу.

Например, чтобы изменить параметры сетевого интерфейса, работающего по протоколу IPv4, можно использовать следующие команды:

- изменение IP-адреса

```
ifconfig [interface alias] [new ip]
```

- изменение маски подсети

```
ifconfig [interface alias] netmask [new netmask]
```

- изменение шлюза

```
ifconfig [interface alias] gateway [new gateway]
```

Согласно модели OSI, сетевые интерфейсы представляют собой канальный и сетевой уровень. Сетевой и транспортный уровень связываются механизмом сокетов. **Socket** – абстрактное программное понятие, используемое для обозначения в прикладной программе конечной точки канала связи с коммуникационной средой, образованной вычислительной сетью. При использовании протоколов TCP/IP или UDP/IP можно говорить, что сокет является средством подключения прикладной программы к порту локального узла сети, где сокет однозначно определяется своим сетевым протоколом (представлен номером из файла /etc/protocols/) и номером порта. Сокет-интерфейс представляет собой набор системных вызовов и/или библиотечных функций языка программирования С, разделенных на четыре группы:

5. локального управления;
6. установления связи;
7. обмена данными (ввода/вывода);
8. закрытия связи.

Для просмотра состояния сокетов из командной строки можно использовать команды **netstat** и **ss**.

Просмотр всех соединений по всем протоколам

```
netstat –a
```

Просмотр только прослушиваемых сокетов

```
netstat –l
```

Просмотр статистики по протоколам

```
netstat –s
```

Для детального рассмотрения всего, что передается по сети, или для учебно-исследовательских целей, чтобы узнать как взаимодействуют между собой объекты сети, рассмотрим команду **tcpdump**.

tcpdump выводит заголовки пакетов проходящих через сетевой интерфейс, которые совпадают с булевым выражением. Он может также быть запущен с ключем -w, который указывает сохранять данные пактов в файл для дальнейшего исследования, и/или с ключем -r, который указывает читать сохраненные пакеты из файла, вместо чтения пакетов из сетевого интерфейса.

tcpdump будет, если не запущен с ключем -c, продолжать собирать пакеты до тех пор, пока не будет прерван сигналом SIGINT или сигналом SIGTERM. Если запуск был с ключем -c, то сбор пакетов будет продолжаться до тех пор, пока не произойдет прерывание сигналом SIGINT или SIGTERM или пока не будет обработано определенное количество пакетов.

Когда **tcpdump** закончит сбор пакетов, то будет сообщено о количестве:

- пакетов "полученных фильтром" (received by filter) (обычно, это пакеты, соответствующие выращению и обработанные командой **tcpdump**);
- пакетов "отброшенных ядром" (dropped by kernel) (это число пакетов, которые были отброшены, в зависимости от механизма сбора пакетов (недостаточного объема буферов) на той ОС, где запускается tcpdump, ОС предоставит эту информацию приложению или нет, и тогда будет выведено число 0)

Рассмотрим фильтрующее выражение. По нему выбираются пакеты из общего потока. Если оно не указано, то будут выбираться и выводится все пакеты, идущие через интерфейс. Иначе, будут обработаны только те пакеты, для которых проверка с выражением выдаст значение «истина».
Выражение состоит из одного или более примитивов. Примитивы обычно состоят из ID (имя или номер) следующего за одним или более классификаторами. Различают три вида классификаторов:

- **type** – вид ID. Возможные значения host, net или port. Пример: 'host foo', 'net 128.3', 'port 20'. Если классификатор **type** не указан, то подразумевается host.
- **dir** – определяет конкретное направление передачи "к" и/или "от" ID. Возможны значения src, dst, src or dst, and src and dst. Пример, 'src foo', 'dst net 128.3', 'src or dst port ftp-data'. Если не указан, то подразумевается src or dst. Для соединений нулевого ('*null*') уровня (к примеру, протокол точка-точка, такой как slip) указанием направления могут быть классификаторы inbound и outbound.
- **proto** – ограничивает совпадение конкретным протоколом. Возможные протоколы: ether, fddi, tr, ip, ip6, arp, rarp, decnet, tcp и udp. Пример, 'ether src foo', 'arp net 128.3', 'tcp port 21'. Если классификатор **proto** не указан, то подразумеваются все перечисленные типы протоколов. Например, 'src foo означает '(ip or arp or rarp) src foo', 'net bar' означает '(ip or arp or rarp) net bar', а 'port 53' означает '(tcp or udp) port 53'.

Также существует несколько специальных примитивов – ключевых слов: gateway, broadcast, less, greater и арифметические выражения.

9.2 Задание для практического занятия

1. Разработать алгоритм и программу согласно варианту задания с использованием языка Си.
2. Проверить функционирование разработанной программы на наборе тестов.
3. Провести анализ этапов выполнения задачи.
4. Подготовить и оформить отчет по выполнению лабораторной работы. Отчет должен содержать титульный лист, техническое задание, блок-схемы алгоритмов, исходный текст программы с комментариями, тесты, результаты их выполнения и выводы по работе.

Порядок выполнения работы

1. Создайте файл исходного текста программы с помощью редактора интегрированной среды разработки **eclipse**.
2. Разработайте алгоритм программы в соответствии с вариантом задания.
3. Разбейте алгоритм на несколько логических блоков.
4. Создайте исходный текст программы, реализующей разработанный алгоритм.
5. Скомпилируйте проект.
6. Устраните синтаксические ошибки, если такие имеются.
7. Разработайте набор тестов с учетом особенностей реализации алгоритма.

8. Проверьте правильность функционирования программы с использованием набора тестов.
9. Устраните семантические ошибки в программе, если такие имеются.

9.3 Задание для самостоятельной работы

1. Программа передачи сообщений в сети (чат) с использованием протокола UDP и отображением имен подключенных клиентов. Предусмотреть подтверждение о доставке пакетов.
2. Программа передачи сообщений в сети (чат) с использованием протокола TCP и отображением имен подключенных клиентов.
3. Передача файлов между двумя станциями сети с использованием протокола UDP, предусмотреть проверку корректности передачи файла, поскольку UDP не обеспечивает гарантированную доставку.
4. Передача файлов между двумя станциями сети с использованием протокола TCP — предусмотреть продолжение передачи (докачку) файла в случае разрыва соединения.
5. Сетевой сканер портов с использование синхронных сокетов. Пользователь должен задавать IP адрес и диапазон портов для проверки.
6. Удаленное управление по протоколу UDP. Клиент посылает команды, которые выполняются на сервере. Результат выполнения команды передается обратно клиенту. Организовать работу с файлами и папками: просмотр директорий удаленного компьютера, копирование, создание, удаление, переименование файлов на удаленном компьютере.
7. Удаленное управление по протоколу TCP. Клиент посылает команды, которые выполняются на сервере. Результат выполнения команды передается обратно клиенту. Организовать работу с файлами и папками: просмотр директорий удаленного компьютера, копирование, создание, удаление, переименование файлов на удаленном компьютере.
8. Тестирование сети (максимальный размер UDP датаграмм, максимальный размер кадра Ethernet, размер входного и выходного буфера передачи).
9. Система агент-менеджер, работающая по протоколу UDP. Программы-агенты собирают сведения о локальных компьютерах (например сведения о конфигурации компьютера: объеме оперативной памяти, частоте процессора, объеме видеопамяти, объеме жесткого диска) и передают их программе-менеджеру при получении запроса от него.
10. Система агент-менеджер, работающая по протоколу TCP. Программы-агенты собирают сведения о локальных компьютерах (например сведения о конфигурации компьютера: объеме оперативной памяти, частоте процессора, объеме видеопамяти, объеме жесткого диска) и передают их программе-менеджеру при получении запроса от него.

11. Программа формирования направленных штормов запросов на указываемые порты. Программа в бесконечном цикле шлет UDP пакеты, не дожидаясь ответа.
12. Программа формирования направленных штормов запросов на указываемые порты. Программа в бесконечном цикле шлет TCP пакеты, не дожидаясь ответа.

9.4 Контрольные вопросы

1. Как отобразить имена подключенных клиентов?
2. Что требуется сделать для работы с файлами и папками, чтобы осуществить просмотр директорий удаленного компьютера?
3. Что требуется сделать для работы с файлами и папками, чтобы осуществить копирование файлов на удаленном компьютере?
4. Что требуется сделать для работы с файлами и папками, чтобы осуществить создание файлов на удаленном компьютере?
5. Что требуется сделать для работы с файлами и папками, чтобы осуществить удаление, переименование файлов на удаленном компьютере?

10 ЛАБОРАТОРНАЯ РАБОТА № 10. СОЗДАНИЕ СКРИПТОВ КОМАНДНОГО ИНТЕРПРЕТАТОРА BASH

Цель работы: научиться создавать скрипты командного интерпретатора bash для обработки файлов.

10.1 Методические рекомендации по выполнению

Для считывания аргументов командной строки используйте следующие переменные:

$# — количество аргументов командной строки (argc – 1);

$@ — строка с аргументами командной строки (без имени скрипта);

$0 — имя запущенного скрипта;

$1, $2, … — аргументы командной строки.

Пример. Написать скрипт, выводящий на экран «битые» символические ссылки.

```
#!/bin/bash

[ $# -eq 0 ] && directorys=`pwd` || directorys=$@

linkchk () {
    for element in $1/*; do
    [ -h "$element" -a ! -e "$element" ] && echo \"$element\"
    [ -d "$element" ] && linkchk $element
    # '-h' проверяет символические ссылки, '-d' -- каталоги.
    done
}

for directory in $directorys; do
    if [ -d $directory ]
        then linkchk $directory
        else
            echo "$directory не является каталогом"
            echo "Порядок использования: $0 dir1 dir2 ..."
    fi
done

exit 0
```

1. Создайте файл исходного текста программы с помощью редактора **nano** или **gedit**.

2. Разработайте алгоритм скрипта в соответствии с вариантом задания.

3. Разбейте алгоритм на несколько логических блоков.

4. Создайте исходный текст скрипта, реализующей разработанный алгоритм.

5. Разрешите доступ к файлу скрипта на исполнение командой chown u+x *script.sh*, где *script.sh* — название скрипта.

6. Устраните синтаксические ошибки, если такие имеются.

7. Разработайте набор тестов с учетом особенностей реализации алгоритма.

8. Проверьте правильность функционирования программы с использованием набора тестов.

9. Устраните семантические ошибки в программе, если такие имеются.

10.2 Задание для практического занятия

1. Разработать алгоритм и скрипт согласно варианту задания с использованием языка командного интерпретатора bash. Скрипт должен принимать аргументы командной строки.
2. Проверить функционирование разработанного скрипта на наборе тестов.
3. Провести анализ этапов выполнения задачи.

Подготовить и оформить отчет по выполнению лабораторной работы. Отчет должен содержать титульный лист, техническое задание, блок-схемы алгоритмов, исходный текст скрипта с комментариями, тесты, результаты их выполнения и выводы по работе.

10.3 Задание для самостоятельной работы

1. Написать программу поиска всех файлов в указанном каталоге, содержащих заданное слово. Формат вызова программы: wordsearch <путь> <искомое слово>. Программа должна выводить имена всех файлов в один столбец.
2. Написать программу поиска файла по имени в указанном каталоге и во всех вложенных. Формат вызова программы: filesearch <путь> <имя файла>. Программа должна вывести путь к искомому файлу.
3. Написать программу вывода списка имен всех файлов из указанного каталога и всех вложенных в алфавитном порядке с указанием размера файла. Формат вызова программы: filelist <путь>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.
4. Написать программу вывода списка имен всех файлов из указанного каталога и всех вложенных в порядке возрастания или убывания размера файла. Формат вызова программы: sizelist <путь> <-a по возрастанию или –d по убыванию>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.
5. Написать программу вывода иерархического списка всех вложенных каталогов, начиная с указанного. Формат вызова программы: catlist <путь>.

Программа должна выводить иерархический список каталогов по строкам: в первой строке — корневой каталог, во второй — его подкаталоги, далее — их подкаталоги и т.д.

6. Написать программу вывода списка имен всех файлов из указанного каталога в алфавитном порядке с группировкой по расширению (список сортируется по возрастанию расширения, внутри каждой группы с одинаковым расширением — в порядке возрастания имени файла) с указанием размера файла. Формат вызова программы: alphalist <путь>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.
7. Написать программу вывода списка имен всех файлов из указанного каталога в алфавитном порядке (по расширению), внутри каждой группы файлов с одинаковым расширением — в порядке возрастания размера файла с указанием размера файла. Формат вызова программы: Xlist <путь>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.
8. Написать программу форматирования файла, содержащего дробные числа. Входные данные — файл с дробными числами в произвольном формате, выходные данные — файл с дробными числами в формате XXX.XX (например, 001.00) в один столбец. Формат вызова программы: floatformat –s <исходный файл> –d <выходной файл>. Если входной или выходной файл не указаны, то данные вводятся с клавиатуры или выводятся на экран.
9. Написать программу автоматического создания и удаления каталогов по заданному шаблону. Шаблон имеет следующий вид: «каталог %d-й», где символ %d должен быть заменен на число. Формат вызова программы: autodir <–c или –d> <шаблон> <начальный номер> <конечный номер>, для создания каталогов с именами cat1, cat2, cat3: autodir –c cat%d 1 3.
10. Написать программу автоматического создания и удаления файлов по заданному шаблону. В каждый файл должно быть записано его имя. Шаблон имеет следующий вид: «file%d.txt», где символ %d должен быть заменен на число. Формат вызова программы: autofile <–c (для создания) или –d (для удаления)> <шаблон> <начальный номер> <конечный номер>, для создания каталогов с именами file1.txt, file2.txt, file3.txt: autodir –c file%d.txt 1 3.
11. Написать программу поиска всех файлов в указанном каталоге, содержащих заданное число. Формат вызова программы: numbersearch <путь> <искомое число>. Программа должна выводить имена всех файлов в один столбец.
12. Написать программу вывода списка имен всех файлов из указанного каталога и всех вложенных в алфавитном порядке (сначала по расширению, затем — по имени) с указанием размера файла. Формат вызова программы: fileextlist <путь>. Программа должна выводить список файлов в три столбца: имя файла, расширение файла, размер файла.

13. Написать программу измерения размера всех подкаталогов в заданном каталоге. Размер подкаталога определяется рекурсивно как сумма размеров всех его файлов, включая файлы во вложенных подкаталогах. Формат вызова программы: dirsize <путь>. Программа должна выводить список каталогов в два столбца: имя каталога, размер каталога.
14. Написать программу переноса всех текстовых файлов из одного каталога в другой с заменой всех букв внутри файла на прописные. Формат вызова программы: mvlowcase <откуда> <куда>. Пример: mvlowcase ~/A1 ~/A2.
15. Написать программу подсчета количества слов для всех текстовых файлов *.txt заданного каталога. Формат вызова программы: wordcount <путь>. Программа должна выводить список файлов в два столбца: имя файла, количество слов в файле.
16. Написать программу вывода иерархического списка всех вложенных каталогов и файлов, начиная с указанного. Формат вызова программы: catfilelist <путь>. Программа должна выводить иерархический список каталогов по строкам: в первой строке — корневой каталог, во второй — его подкаталоги и файлы, далее — их подкаталоги и файлы и т.д.
17. Написать программу автоматического создания символических ссылок на все файлы из указанного каталога и всех его вложенных подкаталогов. Все ссылки должны быть сохранены в одном каталоге. Формат вызова программы: autosymlink <каталог с файлами> <каталог с ссылками>.
18. Написать программу поиска всех файлов в указанном каталоге и во всех вложенных подкаталогах, содержащих в своем имени указанную строку. Формат вызова программы: filenamesearch <путь> <искомая строка>. Программа должна выводить имена всех файлов в один столбец.
19. Написать программу поиска слова наибольшей длины для всех текстовых файлов *.txt заданного каталога. Формат вызова программы: wordlen <путь>. Программа должна выводить список файлов в три столбца: имя файла, найденное слово и его длина.
20. Написать программу автоматического создания символических ссылок на все файлы из указанного каталога, содержащие в названии заданную строку. Все ссылки должны быть сохранены в одном каталоге. Формат вызова программы: filtersymlink <каталог с файлами> <строка> <каталог с ссылками>.
21. Написать программу копирования всех текстовых файлов из одного каталога в другой с заменой всех букв внутри файла на строчные. Формат вызова программы: cplowcase <откуда> <куда>. Пример: cplowcase ~/A1 ~/A2.
22. Написать программу поиска файла по имени в указанном каталоге и во всех вложенных. Формат вызова программы: filesearch <путь> <имя файла>. Программа должна вывести путь к искомому файлу.
23. Написать программу вывода списка имен всех файлов из указанного каталога и всех вложенных в алфавитном порядке с указанием размера фай-

ла. Формат вызова программы: filelist <путь>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.

24. Написать программу вывода списка имен всех файлов из указанного каталога и всех вложенных в порядке возрастания или убывания размера файла. Формат вызова программы: sizelist <путь> <-a по возрастанию или –d по убыванию>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.
25. Написать программу вывода иерархического списка всех вложенных каталогов, начиная с указанного. Формат вызова программы: catlist <путь>. Программа должна выводить иерархический список каталогов по строкам: в первой строке — корневой каталог, во второй — его подкаталоги, далее — их подкаталоги и т.д.
26. Написать программу вывода списка имен всех файлов из указанного каталога в алфавитном порядке с группировкой по расширению (список сортируется по возрастанию расширения, внутри каждой группы с одинаковым расширением — в порядке возрастания имени файла) с указанием размера файла. Формат вызова программы: alphalist <путь>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.
27. Написать программу вывода списка имен всех файлов из указанного каталога в алфавитном порядке (по расширению), внутри каждой группы файлов с одинаковым расширением — в порядке возрастания размера файла с указанием размера файла. Формат вызова программы: Xlist <путь>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.
28. Написать программу форматирования файла, содержащего дробные числа. Входные данные — файл с дробными числами в произвольном формате, выходные данные — файл с дробными числами в формате XXX.XX (например, 001.00) в один столбец. Формат вызова программы: floatformat –s <исходный файл> –d <выходной файл>. Если входной или выходной файл не указаны, то данные вводятся с клавиатуры или выводятся на экран.
29. Написать программу автоматического создания и удаления каталогов по заданному шаблону. Шаблон имеет следующий вид: «каталог %d-й», где символ %d должен быть заменен на число. Формат вызова программы: autodir <–c или –d> <шаблон> <начальный номер> <конечный номер>, для создания каталогов с именами cat1, cat2, cat3: autodir –c cat%d 1 3.
30. Написать программу автоматического создания символических ссылок на все файлы из указанного каталога, содержащие в названии заданную строку. Все ссылки должны быть сохранены в одном каталоге. Формат вызова программы: filtersymlink <каталог с файлами> <строка> <каталог с ссылками>.
31. Написать программу автоматического создания символических ссылок на все файлы из указанного каталога, содержащие в названии заданную

строку. Все ссылки должны быть сохранены в одном каталоге. Формат вызова программы: filtersymlink <каталог с файлами> <строка> <каталог с ссылками>

32. Написать программу автоматического создания и удаления файлов по заданному шаблону. В каждый файл должно быть записано его имя. Шаблон имеет следующий вид: «file%d.txt», где символ %d должен быть заменен на число. Формат вызова программы: autofile <–c (для создания) или –d (для удаления)> <шаблон> <начальный номер> <конечный номер>, для создания каталогов с именами file1.txt, file2.txt, file3.txt: autodir –c file%d.txt 1 3.
33. Написать программу поиска всех файлов в указанном каталоге, содержащих заданное число. Формат вызова программы: numbersearch <путь> <искомое число>. Программа должна выводить имена всех файлов в один столбец.
34. Написать программу вывода списка имен всех файлов из указанного каталога и всех вложенных в порядке возрастания или убывания размера файла. Формат вызова программы: sizelist <путь> <-a по возрастанию или –d по убыванию>. Программа должна выводить список файлов в два столбца: имя файла, размер файла.
35. Написать программу поиска всех файлов в указанном каталоге, содержащих заданное слово. Формат вызова программы: wordsearch <путь> <искомое слово>. Программа должна выводить имена всех файлов в один столбец.

10.4 Контрольные вопросы

1. Для чего служит переменная $#?
2. Для чего служит переменная $@?
3. Для чего служит переменная $0 ?
4. Для чего служит переменная $1, $2,…?

11 ЛАБОРАТОРНАЯ РАБОТА № 11. ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ В LINUX

Цель работы: научиться разрабатывать многозадачные приложения в Linux.

11.1 Методические рекомендации по выполнению

1. Создайте файл исходного текста программы с помощью редактора интегрированной среды разработки **eclipse**.
2. Разработайте алгоритм программы в соответствии с вариантом задания.
3. Разбейте алгоритм на несколько логических блоков.
4. Создайте исходный текст программы, реализующей разработанный алгоритм.
5. Скомпилируйте проект.
6. Устраните синтаксические ошибки, если такие имеются.
7. Разработайте набор тестов с учетом особенностей реализации алгоритма.
8. Проверьте правильность функционирования программы с использованием набора тестов.
9. Устраните семантические ошибки в программе, если такие имеются.

11.2 Задание для практического занятия

1. Разработать алгоритм и программу согласно варианту задания с использованием языка Си. Программа должна принимать аргументы командной строки.
2. Проверить функционирование разработанной программы на наборе тестов.
3. Провести анализ этапов выполнения задачи.
4. Подготовить и оформить отчет по выполнению лабораторной работы. Отчет должен содержать титульный лист, техническое задание, блок-схемы алгоритмов, исходный текст программы с комментариями, тесты, результаты их выполнения и выводы по работе.

11.3 Задание для самостоятельной работы

1. Разработать программу-демона, которая после получения сигнала SIGINT начинает создавать каждую секунду новый процесс-зомби, после получения сигнала SIGTERM удаляет все ранее созданные процессы-зомби.
2. Разработать программу, которая сохраняет список всех файлов из указанного каталога в текстовый файл. Путь к каталогу и имя файла

вводятся с клавиатуры с таймаутом 15 секунд. Если пользователь в течение этого времени не вводит данные, программа принимает их равными по умолчанию «/home/student» и «/home/student/list.txt» соответственно.

3. Разработать программу, состоящую из двух процессов, связанных парой разнонаправленных неименованных каналов. Первый процесс считывает строку с клавиатуры, записывает ее в файл-журнал с префиксом «>:» и передает через неименованный канал во второй процесс. После этого процесс читает строку из второго неименованного канала, сохраняет ее в файл-журнал с префиксом «<:» и выводит на экран. Во втором процессе работает командная оболочка /bin/bash, стандартные потоки ввода и вывода перенаправлены в соответствующие неименованные каналы.
4. Разработать программу, состоящую из двух процессов, связанных парой разнонаправленных неименованных каналов. Первый процесс считывает строку с клавиатуры и передает через неименованный канал во второй процесс. После этого процесс читает строку из второго неименованного канала и выводит на экран. Во втором процессе работает командная оболочка /bin/bash, стандартные потоки ввода и вывода перенаправлены в соответствующие неименованные каналы. Если пользователь вводит команду su или passwd, первый процесс должен зафиксировать введенные пароли в файле-журнале, имя которого указывается в качестве аргумента командной строки.
5. Разработать программу, состоящую из двух процессов: первый процесс считывает данные с клавиатуры, записывает введенные строки в файл-журнал (время ввода и содержимое строки) и передает их через неименованный канал во второй процесс; во втором процессе работает командная оболочка /bin/bash. При этом команды удаления файлов в командную оболочку не передаются. Имя файла-журнала указывается в командной строке при запуске программы.
6. Разработать простейшую командную оболочку, запускающую вводимые в командной строке команды на выполнение и ожидающую их завершения. По завершении команды командная оболочка должна выдавать приглашение вида: (12:00:05 /home/student :-) _. Если пользователь в течение 30 секунд не вводит команду, командная оболочка снова выводит приглашение на новой строке.
7. Разработать программу параллельного вычисления произведения двух квадратных матриц N×N четырьмя процессами. Матрицы считываются из файлов, указываемых в командной строке при запуске программы. Дочерним процессам передаются имена файлов и диапазоны вычисляемых каждым процессом строк. Дочерние процессы возвращают вычисленные строки результирующей матрицы. Обмен производится с помощью неименованного канала.

8. Разработать программу параллельного вычисления суммы ряда Тейлора $1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+...+\frac{x^k}{k!}+...$ Первый дочерний процесс вычисляет сумму нечетных членов, второй — сумму четных. Вычисления проводятся до достижения точности 2*DBL_EPSILON (см. float.h). Результат вычисления передается в родительский процесс через неименованный канал.
9. Разработать программу-демона, которая после получения сигнала SIGUSR1 начинает создавать каждую секунду просматривать содержимое каталога и при появлении нового файла создает на него жесткую ссылку и сохраняет ее в этом же каталоге с добавлением к имени файла расширения «.bak». При получения сигнала SIGUSR2 программа удаляет все жесткие ссылки с расширением .bak. Путь к просматриваемому каталогу указывается в командной строке.
10. Разработать программу-демона, которая после получения сигнала SIGINT начинает создавать каждую секунду просматривать содержимое текстового файла, и если он начинается не со строки «Это вирус», добавляет эту строку в начало файла. При получения сигнала SIGTERM программа удаляет строку «Это вирус» из файла. Путь к файлу указывается в командной строке.
11. Разработать программу-демона, состоящую из двух процессов. Первый процесс раз в секунду фиксирует размер текстового файла и при его изменении через неименованный канал передает новый размер файла второму процессу, который, в свою очередь, записывает время изменения и новый размер файла в файл-журнал. Имена текстового файла и файла-журнала указываются в командной строке.
12. Разработать программу-чат между пользователями разных терминалов одного компьютера. Взаимодействие осуществляется с помощью именованных каналов FIFO и сигналов.
13. Разработать программу-чат между пользователями разных терминалов одного компьютера. Взаимодействие осуществляется с помощью сообщений.
14. Разработать программу-чат между пользователями разных терминалов одного компьютера. Взаимодействие осуществляется с помощью разделяемой памяти и семафоров.
15. Разработать программу-чат между пользователями разных терминалов одного компьютера. Взаимодействие осуществляется с помощью локальных сокетов.
16. Разработать игру крестики-нолики для пользователей разных терминалов одного компьютера. Взаимодействие осуществляется с помощью сообщений.

17. Разработать программу-шпиона за действиями пользователя на другом терминале компьютера. Данная программа должна состоять из двух процессов, связанных неименованными каналами. Первый процесс считывает данные с клавиатуры, записывает введенные строки в именованный канал FIFO и передает их через другой неименованный канал во второй процесс. Во втором процессе работает командная оболочка /bin/bash, стандартный вывод которой направляется в первый процесс и отображается пользователю и одновременно записывается в именованный канал FIFO. Для чтения именованного канала FIFO можно использовать команду cat.
18. Разработать программу параллельного вычисления произведения двух квадратных матриц N×N четырьмя процессами. Матрицы считываются из файлов, указываемых в командной строке при запуске программы. Дочерним процессам передаются двумерные массивы и диапазоны вычисляемых каждым процессом строк. Дочерние процессы возвращают вычисленные строки результирующей матрицы. Массивы передаются помощью разделяемой памяти.
19. Разработать программу-демона, которая раз в секунду подсчитывает размер всех файлов в каталоге A и если он превышает заданный, то переносит самый большой файл в каталог B, отмечая при этом в журнале c.log время переноса и имя перенесенного файла и заменяя этот файл символической ссылкой на него. Управление демоном (запуск, останов, задание параметров A, B, c.log) осуществляется с помощью сервисной программы, взаимодействующей с ним с помощью именованных каналов и сигналов.
20. Разработать программу-демона, которая один раз в секунду просматривает содержимое каталога A и при появлении нового файла, например, с именем filename.ext создает новую скрытую папку с именем filename.ext.dir, перемещает в нее этот файл, а вместо него оставляет относительную символическую ссылку. Управление демоном (запуск, останов, задание параметра A) осуществляется с помощью сервисной программы, взаимодействующей с ним с помощью сообщений.
21. Разработать программу-демона, которая один раз в секунду просматривает содержимое каталога A и при появлении нового файла удаляет его, при этом отмечая в специальном файле b.log время удаления и имя удаленного файла. Управление демоном (запуск, останов, задание параметров A и b.log) осуществляется с помощью сервисной программы, взаимодействующей с ним с помощью разделяемой памяти и семафоров.
22. Разработать программу-демона, которая один раз в секунду просматривает содержимое каталога A и при появлении нового файла создает на него жесткую ссылку и сохраняет ее в этом же каталоге с добавлением к имени файла расширения «.bak». Управление демоном (запуск, останов, задание параметра A) осуществляется с помощью

сервисной программы, взаимодействующей с ним с помощью именованных каналов и сигналов.

23. Разработать программу-демона, которая один раз в секунду фиксирует размер текстового файла a.txt и при его изменении записывает сведения в журнал b.log (время изменения и новый размер). Управление демоном (запуск, останов, задание параметра A) осуществляется с помощью сервисной программы, взаимодействующей с ним с помощью именованных каналов и сигналов.

11.4 Контрольные вопросы

1. Как фиксировать размер текстового файла a.txt?
2. Что необходимо использовать для учета таймаутом 10 секунд ?
3. Что требуется сделать для просмотра содержимого текстового файла?
4. С помощью чего считываются данные из файлов, указываемых в командной строке при запуске программы?

12 КРАТКИЙ СПРАВОЧНИК КОМАНД LINUX ПО КАТЕГОРИЯМ

12.1 Получение справки

apropos

Назначение

Просмотр списка всех man-страниц, содержащих указанное ключевое слово.

Синтаксис

```
apropos ключевое_слово
```

Опции Нет

Описание

Команда `apropos` ищет ключевое слово в базе данных (называемой базой данных `whatis`), создаваемой программой `/usr/sbin/makewhatis`. База данных `whatis` является индексом ключевых слов, содержащихся во всех man-страницах системы..

info

Назначение

Выводит оперативную справку по любой команде Linux.

Синтаксис

```
info [опции] команда
```

Опции

`-d` каталог добавляет каталог в список каталогов, в которых ведется поиск файлов.

`-f файл_инфо` указывает файл, используемый командой `info`.

`-h` выводит информацию по применению `info`.

Описание

Команда `info` выводит оперативную справку по указанной команде в полноэкранном текстовом окне. Дополнительные сведения о команде `info` можно получить, набрав `info` без аргументов.

man

Назначение

Выводит страницы оперативного руководства (называемые также man-страницами).

Синтаксис

```
man [опции] [раздел] команда
```

Опции

`-C конф_файл` указывает конфигурационный файл `man` (по умолчанию `/etc/man.config`).

`-P листатель` указывает программу постраничного вывода руководства (например, `less`).

`-a` выводит все man-страницы, соответствующие конкретной команде.

`-h` выводит только справку по команде `man`.

`-w` показывает расположение отображаемых man-страниц.

Описание

Команда man выводит man-страницы для указанной команды. Если вам известен раздел man-

страниц, вы можете указать и его.

whatis

Назначение

Производит поиск целых слов в базе данных `whatis` (см. команду `apropos`).

Синтакси

```
whatis ключевое_слово
```

Опции

Нет

Описание

Команда whatis выводит результат поиска целых слов в базе данных whatis (см. описание команды apropos). Отображаются только совпадения с целыми словами.

12.2 Облегчение ввода команд

alias

Назначение

Определение аббревиатуры для длинной команды или просмотр текущего списка аббревиатур.

Синтаксис

```
alias [abbrev=команда]
```

Опции

Нет

Описание

Если вы введете только alias, то получите список всех определенных на данный момент аббревиатур. Обычно команда alias используется для определения легко запоминаемых аббревиатур для более длинных команд. Например, если вы часто набираете команду ls -l, можете добавить в файл .bashrc в домашнем каталоге строку с командой alias ll='ls -l'. Затем для просмотра подробного оглавления каталога вместо команды ls -l можно вводить команду ll. alias является встроенной командой командного процессора Bash.

unalias

Назначение

Уничтожает аббревиатуру, определенную ранее с помощью команды alias.

Синтаксис

```
unalias аббревиатура
```

Опции

Нет

Описание

Команда unalias удаляет аббревиатуру, определенную ранее с помощью команды alias.

Команда unalias является встроенной командой командного процессора Bash.

12.3 Управление файлами и каталогами

cd

Назначение

Изменяет текущий каталог.

Синтаксис

```
cd [каталог]
```

Опции

Нет

Описание

Команда `cd` без имени каталога изменяет текущий каталог на домашний каталог пользователя. В противном случае `cd` изменяет каталог на указанный в команде. Команда `cd` является встроенной командой командного процессора Bash.

chgrp

Назначение

Изменяет группового владельца одного или нескольких файлов.

Синтаксис

```
chgrp [-cfvR] группа файлы
```

Опции

`-c` выводит только файлы с измененным групповым владельцем.

`-f` подавляет вывод сообщений об ошибках.

`-v` подробно сообщает об изменениях группового владельца.

`-R` рекурсивно изменяет группового владельца файлов во всех подкаталогах.

Описание

Для изменения группового владельца одного или более файлов введите `chgrp` с именем группы, а за ним – имена файлов (группового владельца можно также изменить и с помощью команды `chown`).

chmod

Назначение

Изменяет права доступа одного или нескольких файлов.

Синтаксис

```
chmod [-cfvR] права_доступа файлы
```

Опции

`-c` выводит только файлы с измененными правами доступа.

`-f` подавляет вывод сообщений об ошибках.

`-v` выдает подробную информацию об изменениях прав доступа.

`-R` рекурсивно изменяет права доступа файлов во всех подкаталогах.

Описание

Для эффективного применения `chmod` необходимо знать, как задавать права доступа. Один из способов предполагает конкатенацию по одной букве из каждой из следующих таблиц в порядке их следования (Кто/Действие/Доступ):

Кто	Действие	Доступ
`u` пользователь	`+` добавить	`r` чтение
`g` группа	`-` удалить	`w` запись
`o` другие	`=` назначить	`x` выполнить
`a` все	`s` установить ID пользователя	

Чтобы предоставить доступ на чтение ко всем файлам каталога, введите `chmod a+r *`. А для разрешения выполнения указанного файла всем пользователям наберите `chmod +x имя_файла`. Другим способом указания прав доступа является использование последова- тельности из трех восьмеричных цифр. В подробном листинге значения прав на чтение, запись и выполнение для пользователя, группы и других выглядят как последовательность `rwxrwxrwx` (с прочерками на месте букв для запрещенных операций). Строку `rwxrwxrwx` можно рассматривать как трехкратное повторение строки `rwx`. Теперь присвойте значения `r=4`, `w=2` и `x=1`. Чтобы получить значение последовательности `rwx`, сложите значения `r`, `w` и

`x`. Таким образом, `rwx` = 7. С помощью этой формулы вы можете присвоить трехзначное значение любым правам доступа.

chown

Назначение

Изменяет владельца или группового владельца файла.

Синтаксис

```
chown [cvfR] имя_пользователя:имя_группы файлы
```

Опции

`-c` выводит только файлы с измененными владельцами.

`-f` подавляет вывод сообщений об ошибках.

`-v` выдает подробную информацию об изменениях владельцев.

`-R` рекурсивно изменяет владельцев файлов во всех подкаталогах.

Описание

Чтобы сделать пользователя владельцем одного или более файлов, введите команду `chown` с именем пользователя, а за ним – именами файлов. Для из-

менения группового владельца, добавьте к имени пользователя после точки новое имя группы.

cp

Назначение

Копирует файлы и каталоги.

Синтаксис

```
cp [опции] исходный_файл файл_назначения
cp [опции] исходные_файлы каталог_назначения
```

Опции

`-a` сохраняет все атрибуты файла.

`-b` создает перед копированием резервную копию файла.

`-d` копирует ссылку, но не файл, на который указывает эта ссылка.

`-i` запрашивает подтверждение перед перезаписью файлов.

`-l` создает жесткие ссылки вместо копирования файлов.

`-p` сохраняет владельцев, права доступа и метку времени файла.

`-R` рекурсивно копирует файлы во всех подкаталогах.

`-s` создает мягкие ссылки вместо копирования файлов.

`-u` копирует файл только в том случае, если он более новый, чем файл назначения.

`-v` выводит подробные сообщения во время процесса копирования.

`-help` выводит справку по команде.

Описание

Команда `cp` копирует один файл в другой. Можно также копировать несколько файлов из одного каталога в другой.

ln

Назначение

Устанавливает жесткие или символические ссылки (псевдонимы) для файлов и каталогов.

Синтаксис

```
ln [опции] существующий_файл новое_имя
```

Опции

`-b` выполняет резервное копирование файлов перед их удалением.

`-d` создает жесткую ссылку на каталог (это может сделать только `root`).

`-f` удаляет существующий файл с именем `новое_имя`.

`-help` выводит справку по команде.

`-s` создает символическую ссылку.

`-v` формирует подробную выходную информацию.

Описание

Команда `ln` назначает новое имя существующему файлу. С помощью опции `-s` можно создать символические ссылки, известные во всех файловых системах. Для символической ссылки можно просмотреть связанную с ней информацию с помощью команды `ls -l`. В противном случае `ls -l` выводит два отдельных файла – для файла и его жесткой ссылки.

ls

Назначение

Выводит оглавление каталога.

Синтаксис

`ls [опции] [имя_каталога]`

Опции

`-a` выводит все файлы, в том числе и с именами, начинающимися с точки (.).

`-b` выводит непечатные символы в именах файлов в восьмеричном коде.

`-c` сортирует файлы по времени их создания.

`-d` выводит каталоги так же, как и обычные файлы (а не выводит их оглавление).

`-f` выводит оглавление каталога без сортировки (именно так, как оно хранится на диске).

`-i` выводит информацию об индексных дескрипторах inode.

`-l` выводит список файлов в длинном формате с подробной информацией.

`-p` добавляет к имени файла символ, отображающий его тип.

`-r` сортирует оглавление в обратном алфавитном порядке.

`-s` выводит рядом с именами файлов их размер (в килобайтах).

`-t` сортирует оглавление по меткам времени файлов.

`-1` выводит список имен файлов в один столбец.

`-R` рекурсивно выводит файлы во всех подкаталогах.

Описание

Команда `ls` выводит оглавление указанного каталога. Если имя каталога опущено, `ls` выводит оглавление текущего каталога. По умолчанию `ls` не отображает файлы, имена которых начинаются с точки (.); чтобы увидеть все файлы, наберите `ls -a`. Подробную информацию о файлах (включая размер, владельца и группового владельца и права на чтение

– запись – выполнение) можно получить с помощью команды `ls -l`.

mkdir

Назначение

Создает каталог.

Синтаксис

```
mkdir [опции] имя_каталога
```

`-m` доступ назначает новому каталогу указанные права доступа.

`-p` создает родительские каталоги, если они не существуют.

Описание

Команда `mkdir` создает указанный каталог.

mv

Назначение

Переименовывает файлы и каталоги или перемещает их из одного каталога в другой.

Синтаксис

```
mv [опции] источник назначение
```

Опции

`-b` создает резервные копии файлов, которые перемещаются или переименовываются.

`-f` удаляет существующие файлы, не запрашивая подтверждение.

`-i` запрашивает подтверждение перед перезаписью существующих файлов.

`-v` выводит имя файла перед его перемещением.

Описание

Команда `mv` либо переименовывает файл, либо перемещает его в другой каталог. Команда работает как с обычными файлами, так и с каталогами. Таким образом, с помощью команды `mv sample sample.old` вы можете переименовать файл `sample` в `sample.old`. С другой стороны, с помощью команды `mv /tmp/sample /usr/local/sample` вы можете переместить файл `sample` из каталога `/tmp/` в каталог `/usr/local/`.

pwd

Назначение

Выводит текущий рабочий каталог.

Синтаксис

```
pwd
```

Опции

Нет

Описание

Команда `pwd` выводит текущий рабочий каталог. Она является встроенной командой командного процессора Bash.

rm

Назначение

Удаляет один или более файлов.

Синтаксис

```
rm [опции] файлы
```

Опции

`-f` удаляет файлы без запроса подтверждения.

`-i` запрашивает подтверждение перед удалением файлов.

`-r` рекурсивно удаляет файлы во всех подкаталогах, содержащихся в каталоге.

`-v` выводит имена файлов перед их удалением.

Описание

Команда `rm` удаляет указанные файлы. Для удаления файла вы должны иметь право на запись в каталог, содержащий этот файл.

rmdir

Назначение

Удаляет указанный каталог (при условии, что он пуст).

Синтаксис

```
rmdir [опции] каталог
```

Опции

`-p` удаляет все становящиеся пустыми родительские каталоги.

Описание

Команда `rmdir` удаляет пустые каталоги. Если каталог не пуст, необходимо удалить все файлы вместе с каталогом с помощью команды `rm -r`.

touch

Назначение

Изменяет метку времени файла.

Синтаксис

```
touch [опции] файлы
```

Опции

`-c` предписывает `touch` не создавать файл, если он не существует.

`-d` время использует указанное время.

`-r` файл использует отметку времени указанного файла.

`-t ММДДччмм[[ВВ]ГГ][.сс]` использует указанную дату и время.

Описание

Команда `touch` позволяет изменить дату и время последней модификации файла (эта информация хранится вместе с файлом). При вводе `touch` без опций в качестве метки времени файла используются текущие дата и время. Если указанный файл не существует, `touch` создает новый файл размером 0 байт.

12.4 Поиск файлов

find

Назначение

Выводит список файлов, удовлетворяющих заданному набору критериев.

Синтаксис

`find [путь] [опции]`

Опции

`-depth` обрабатывает сначала текущий каталог, затем его подкаталоги.

`-maxdepth` *n* ограничивает поиск *n* уровнями вложенности каталогов.

`-follow` обрабатывает каталоги, указанные символическими ссылками.

`-name` шаблон находит файлы с именами, соответствующими шаблону.

`-ctime` *n* сравнивает файлы, модифицированные точно *n* дней назад.

`-user` имя находит файлы, владельцем которых является указанный пользователь.

`-group` имя находит файлы, владельцем которых является указанная группа.

`-path` шаблон находит файлы, с путями, соответствующими шаблону.

`-perm` права находит файлы с заданными правами доступа.

`-size+`*n*`K` находит файлы с размером более *n* килобайт.

`-type` *x* находит файлы заданного типа, где *x* – один из следующих типов:

`f` сравнивает файлы,

`d` сравнивает каталоги,

`l` сравнивает символические ссылки.

`-print` выводит имена найденных файлов.

`-exec команда [опции] {} \;` выполняет указанную команду, передав ей имя найденного файла.

Описание

Команда `find` удобна для поиска всех файлов, удовлетворяющих заданному набору критериев. Если ввести `find` без аргументов, будет выведен список всех файлов во всех подкаталогах текущего каталога. Для отображения всех файлов с именами, заканчивающимися на `.gz`, введите `find . -name "*.gz"`

locate

Назначение

Выводит все файлы из периодически обновляемой базы данных, которые соответствуют заданному шаблону.

Синтаксис

`locate шаблон`

Опции

Нет

Описание

Команда `locate` производит поиск в базе данных файлов тех имен, которые удовлетворяют указанному `шаблону`. Ваша Linux–система настроена на периодическое обновление базы данных файлов. Если вы не уверены, где находится файл, просто введите `locate`, а за ним часть имени файла.

whereis

Назначение

Производит поиск исходного текста, двоичного файла и man-страницы для команды.

Синтаксис

`whereis [опции] команда`

Опции

`-b` производит поиск только двоичных файлов.

`-m` производит поиск только man-страниц.

`-s` производит поиск только исходных текстов.

Описание

Команда `whereis` производит поиск в обычных каталогах (где находятся двоичные файлы, man-страницы и исходные файлы) двоичных файлов, man-страниц и исходных файлов для заданной команды.

which

Назначение

Производит поиск заданной команды в каталогах, перечисленных в переменной среды `PATH`.

Синтаксис

`which команда`

Опции

Нет

Описание

Команда `which` производит поиск в каталогах, перечисленных в переменной среды `PATH`, файла, который запускается в результате ввода указанной команды. Это удобный способ проверки, что именно выполняется при вводе конкретной команды.

12.5 Работа с файлами

cut

Назначение

Копирует выбранные части каждой строки текстового файла на стандартное устройство вывода.

Синтаксис

`cut [опции] файл`

Опции

`–b` список выбирает символы в позициях, указанных в списке список.

`–f` список выбирает поля (разделенные символами табуляции), указанные в списке список.

`–d` символ указывает символ–разделитель полей (по умолчанию это символ табуляции).

`–s` пропускает строки, не содержащие полей с разделителями (см. опцию –f).

Описание

Команда `cut` выбирает указанные части из каждой строки текстового файла и выводит эти строки на стандартное устройство вывода. Из каждой строки можно выбирать либо диапазон символов (указанный их позициями), либо заданные поля, при этом поля разделяются специальным символом, наподобие символа табуляции.

cmp

Назначение

Выполняет побайтовое сравнение двух файлов.

Синтаксис

`cmp [опции] файл`*`1`* `файл`*`2`*

Опции

`-l` выводит номер байта (десятичный) и различающиеся байты (восьмеричные) для каждого несовпадения.

`-s` возвращает только код завершения (0 – идентичные файлы; 1 – различные файлы; 2 –
недоступный или пропущенный аргумент).

Описание

Команда `cmp` сравнивает `файл1` и `файл2`. Если `файл1` заменен знаком '–', используется стандартный ввод. По умолчанию команда `cmp` не выдает никаких сообщений, если файлы совпадают; если файлы отличаются, выдаются позиция в строке и номер строки, в которой находится первый несовпадающий байт. Если один из файлов является началом другого, то выдается сообщение:

```
cmp:  EOF on имя_более_короткого_файла
```

cut

Назначение

Копирует выбранные части каждой строки текстового файла на стандартное устройство вывода.

Синтаксис

```
cut [опции] файл
```

Опции

`-b` список выбирает символы в позициях, указанных в списке список.

`-f` список выбирает поля (разделенные символами табуляции), указанные в списке список.

`-d` символ указывает символ–разделитель полей (по умолчанию это символ табуляции).

`-s` пропускает строки, не содержащие полей с разделителями (см. опцию –f).

Описание

Команда `cut` выбирает указанные части из каждой строки текстового файла и выводит эти строки на стандартное устройство вывода. Из каждой строки можно выбирать либо диапазон символов (указанный их позициями), либо заданные поля, при этом поля разделяются специальным символом, наподобие символа табуляции.

diff

Назначение

Показывает отличия между двумя текстовыми файлами (или для всех файлов с одинаковыми именами в двух разных каталогах).

Синтаксис

```
diff [опции] первый_файл второй_файл
```

Опции

`-a` считает все файлы текстовыми, даже если это не так.

`-b` игнорирует пустые строки и последовательности пробелов.

`-c` формирует выходные данные в другом формате.

`-d` пытается найти минимальный набор изменений (это существенно замедляет работу `diff`).

`-e` формирует сценарий редактора `ed` для преобразования файла `первый_файл` во

`второй_файл`.

`-f` формирует выходные данные, аналогичные –e, но в обратном порядке.

`-i` игнорирует регистр символов.

`-l` передает выходные данные команде `pr` для разбивки на страницы.

`-n` похоже на `-f`, но подсчитывает количество измененных строк.

`-r` рекурсивно сравнивает файлы с одинаковыми именами во всех подкаталогах.

`-s` сообщает о том, что файлы совпадают.

`-t` заменяет в выходных данных символы табуляции на пробелы.

`-u` использует унифицированный формат вывода.

`-v` выводит версию `diff`.

`-w` при сравнении строк игнорирует пробелы и символы табуляции.

Описание

Команда `diff` сравнивает файлы `первый_файл` и `второй_файл` и выводит различающиеся строки.

dos2unix

Назначение

Преобразует формат текстового файла DOS/Mac к UNIX формату.

Синтаксис

```
dos2unix [опции] [-o файл ...] [-n исходный_файл файл_назначения
...]
```

Опции

`-h` выдает справку о команде.

`-k файл_назначения` сохраняет дату создания исходного файла.

`-q` подавляет сообщения и предупреждения в ходе выполнения команды.

`-V` выводит номер версии.

`-c` устанавливает режим преобразования: ASCII (по умолчанию), 7bit, ISO, Mac.

Описание

Команда `dos2unix` используется для преобразования форматов текстовых файлов DOS/Mac

к UNIX формату. Известно, что форматы текстовых файлов в разных операционных системах

отличаются. Так строки тестовых файлов в DOS заканчиваются парой символов CR и LF (возврат каретки и перевод строки), в UNIX - только символом LF. В режиме `-o` (работает по умолчанию) все файлы, указанные в командной строке, переписываются в формате UNIX. В режиме `-n` отдельно указываются `исходный_файл` и `файл_назначения`.

file

Назначение

Выводит тип данных файла на основе правил, определенных в файле `/usr/lib/magic`

(известного под названием магического файла (*magicfile*)).

Синтаксис

```
file [опции] файлы
```

Опции

`-c` выводит в сформатированном виде указанный магический файл (или файл по умолчанию) и завершает работу.

`-m файл`*1* [: `файл`*2* : ...] указывает другие магические файлы.

`-z` просматривает сжатые файлы.

Описание

Для определения типа данных в указанных файлах команда `file` использует правила, заданные в файле `/usr/lib/magic`. Например, с помощью команды `file` вы можете проверить тип каждого файла в каталоге `/usr/lib` следующим образом:

```
file * | more
```

grep

Назначение

Осуществляет поиск в одном или более файлах строк, соответствующих регулярному выражению (шаблону поиска).

Синтаксис

`grep [опции] шаблон файлы`

Опции

`-N` (где *N*– число) выводит *N* строк вблизи строки, содержащей образец.

`-c` выводит количество строк, содержащих образец поиска.

`-f` файл читает опции из указанного файла.

`-i` игнорирует регистр букв.

`-l` выводит имена файлов, содержащих образец.

`-n` выводит номера строк рядом со строками, содержащими образец.

`-q` возвращает код состояния, но ничего не выводит.

`-v` выводит строки, не содержащие образец.

`-w` сравнивает только целые слова.

Описание

Команда `grep` осуществляет поиск шаблона в указанных файлах. Обычно команда `grep` используется для поиска заданной последовательности символов в одном или нескольких текстовых файлах.

head

Назначение

Отображает первые несколько строк файла на стандартный вывод.

Синтаксис

`head [опции] файл`

Опции

`-n` *n*`строк` количество выводимых строк.

Описание

Выводит на экран первые строки указанного файла `файл`. По умолчанию число строк равно

10. Число выводимых строк можно указать с использованием опции `-n`.

less

Назначение

Поэкранно отображает текстовые файлы (с возможностью листания назад).

Синтаксис

`less [опции] имена_файлов`

Опции

`-?` выводит список команд, которые можно использовать в `less`.

`-p` текст выводит первую строку, в которой найден текст.

`-s` сжимает несколько пустых строк в одну пустую строку.

Описание

Команда `less` поэкранно отображает указанные файлы. В отличие от `more`, файл можно листать назад с использованием клавиш <b>, <Ctrl+b> или <Esc+V>. Для просмотра команд управления `less` при просмотре файла с помощью `less` нажмите <h>.

more

Назначение

Выполняет поэкранный просмотр текстовых файлов.

Синтаксис

```
more [опции] имена_файлов
```

Опции

`+`*N* (где *N* – число) выводит файл, начиная с указанного номера строки.

`+ /шаблон` начинает с отображения двух строк перед шаблоном.

`-s` выводит вместо нескольких пустых строк одну пустую строку.

Описание

Команда `more` выполняет поэкранное отображение указанных файлов. Для просмотра команд, которые можно использовать в `more`, нажмите <h> во время просмотра файла. Для более удобного просмотра файла применяйте команду `less`.

tail

Назначение

Выводит несколько последних строк файла.

Синтаксис

```
tail [опции] файл
```

Опции

`-`*N* (где *N* – число) выводит последние *N* строк.

`-n` *N* (где *N* – число) выводит последние *N* строк.

`-f` читает файл через заданные промежутки времени и выводит все новые строки.

Описание

Команда `tail` выводит последние строки указанного файла. По умолчанию отображаются последние 10 строк файла.

wc

Назначение

Выводит количество байт, слов и строк, находящихся в файле.

Синтаксис

```
wc [опции] [файлы]
```

Опции

`-c` выводит только количество байт.

`-w` выводит только количество слов.

`-l` выводит только количество строк.

Описание

Команда `wc` выводит количество байт, слов и строк, находящихся в файле. Если входной файл не указан, `wc` читает данные из стандартного устройства ввода.

zcat, zless, zmore

Назначение

Просмотр содержимого сжатого текстового файла без его распаковки.

Синтаксис

```
zcat имя_файла zless имя_файла zmore имя_файла
```

Опции

Нет

Описание

Команды `zcat`, `zless` и `zmore` выполняются так же, как и команды `cat`, `less` и `more`. Единственное отличие состоит в том, что `z`–команды могут непосредственно читать `zip`- файлы (без предварительной их распаковки с помощью команды `gunzip`). Эти команды особенно удобны для чтения сжатых текстовых файлов.

12.6 Управление процессами

kill

Назначение

Посылает сигнал процессу.

Синтаксис `kill [опции] id_процесса`

Опции

–*N*сигнала (где *N*сигнала – номер или имя) посылает указанный сигнал.

–l выводит номера и имена сигналов.

Описание

Команда kill посылает сигнал процессу. Сигналом по умолчанию для kill является сигнал

TERM Обычно этот сигнал предназначен для завершения процесса. Например, kill –9

123 завершает выполнение процесса с идентификатором 123. Для просмотра идентификаторов процессов воспользуйтесь командой ps. Для просмотра списка имен и номеров сигналов введите kill –l.

killall

Назначение

Уничтожает все активные процессы.

Синтаксис

killall [опции]

Опции

см. опции команды kill

Описание

Команда killall уничтожаетвсе активные процессы. Подразумеваемое значение посылаемого сигнала равно 9 (сигналу уничтожения).

ldd

Назначение

Выводит имена совместно используемых библиотек, требуемых для выполнения программы.

Синтаксис

ldd [опции] программы

Опции

–v выводит номер версии ldd.

–V выводит номер версии динамического компоновщика (ld.so),

–d перераспределяет функции и сообщает о недостающих функциях.

–r перераспределяет и данные, и функции, и сообщает о недостающих объектах.

Описание

Команда ldd позволяет определить, какие совместно используемые библиотеки требуются для выполнения указанных программ.

ps

Назначение

Отображает состояние процессов (программ), выполняющихся в системе.

Синтаксис

```
ps [опции]
```

Обратите внимание, что в отличие от других команд опции команды `ps` не имеют префикса
"–".

`a` отображает процессы других пользователей.

`f` отображает дерево процессов.

`j` выводит выходные данные, используя формат заданий.

`l` выводит данные в длинном формате, с детальной информацией о каждом процессе.

`m` выводит информацию об использовании памяти каждым процессом.

`u` выводит имя пользователя и время запуска.

`x` выводит процессы, не связанные ни с каким терминалом.

Описание

Команда `ps` отображает состояние процессов, выполняющихся в системе. Команда `ps` без параметров формирует список процессов, запущенных вами. Для просмотра списка всех процессов, выполняемых в системе, введите `ps ax` (или `ps aux`, если вам нужна подробная информация по каждому процессу).

pstree

Назначение

Отображает все выполняющиеся процессы в виде дерева.

Синтаксис

```
pstree [опции] [id_процесса]
```

Опции

`-a` выводит аргументы командной строки.

`-c` не сжимает поддеревья.

`-l` выводит длинные линии (для дерева).

`-n` сортирует процессы по их идентификаторам (а не по именам).

`-p` выводит идентификаторы процессов.

Описание

Команда `pstree` отображает все процессы в форме дерева – так легче увидеть отношения предшествования процессов.

top

Назначение

Выводит список выполняющихся в данный момент процессов, упорядоченных по доле использования процессорного времени.

Синтаксис

```
top [q] [d интервал]
```

Опции

`q` предписывает выполнение `top` с максимально возможным приоритетом (для этого вы должны быть привилегированным пользователем).

`d интервал` указывает интервал в секундах между обновлениями информации.

Описание

Команда `top` формирует полноэкранный отчет о выполняемых процессах с учетом их доли использования процессорного времени. По умолчанию `top` обновляет информацию каждые 5 секунд. Для прекращения выполнения `top` нажмите <q> или <Ctrl–C>.

12.7 Архивирование и сжатие файлов

gunzip

Назначение

Распаковывает `zip`-файлы.

Синтаксис

```
gunzip [опции] файлы
```

Опции

См. опции для `gzip`.

Описание

Команда `gunzip` распаковывает сжатые файлы (имеющие расширение `.gz` или `.Z`). После распаковки `gunzip` заменяет сжатые файлы их распакованными версиями и удаляет из имен файлов расширение `.gz` или `.Z`. Команда `gunzip` эквивалентна команде `gzip` с опцией `–d`.

gzip

Назначение

Сжимает один или более файлов.

Синтаксис

```
gzip [опции] файлы
```

Опции

`-c` выводит выходной файл на стандартное устройство вывода и сохраняет исходный файл.

`-d` распаковывает файл (то же самое, что и `gunzip`).

`-h` выводит справку по команде.

`-l` выводит содержание сжатого файла.

`-n` не сохраняет исходное имя и метку времени.

`-r` рекурсивно сжимает файлы во всех подкаталогах.

`-v` формирует подробные выходные данные.

`-V` выводит номер версии.

Описание

Команда `gzip` сжимает файлы с помощью алгоритма сжатия Лемпеля–Зива (Lempel–Ziv) LZ77. После сжатия файла `gzip` заменяет исходный файл его сжатой версией и добавляет к имени файла `.gz`.

tar

Назначение

Создает архив файлов или извлекает файлы из архива.

Синтаксис

```
tar [опции] файлы_или_каталоги
```

Опции

`-c` создает новый архив.

`-d` сравнивает файлы из архива с файлами из текущего каталога.

`-r` добавляет файлы в архив.

`-t` выводит оглавление архива.

`-x` извлекает файлы из архива.

`-C каталог` извлекает файлы в указанный каталог.

`-f файл` читает архив не с ленты, а из указанного файла.

`-L n` определяет емкость ленты равной *n* килобайт.

`-N дата` архивирует только файлы новее указанной даты.

`-T файл` архивирует или извлекает файлы с именами, указанными в файле `файл`.

`-v` выводит подробные сообщения.

`-z` сжимает или распаковывает архив с помощью `gzip`.

`-j` сжимает или распаковывает архив с помощью `bzip2`.

Описание

Команда `tar` создает архив файлов или извлекает файлы из существующего архива.

12.8 Управление пользователями

groups

Назначение

Показывает группы, которым принадлежит пользователь.

Синтаксис

```
groups [имя_пользователя]
```

Опции

Нет

Описание

Команда `groups` выводит имена групп, к которым принадлежит пользователь. Если не указать имя пользователя, то команда выведет группы текущего пользователя.

id

Назначение

Выводит идентификатор пользователя, идентификатор группы и группы пользователя.

Синтаксис

```
id [опции] [имя_пользователя]
```

Опции

`-g` выводит только идентификатор группы.

`-n` выводит имя группы, а не идентификатор группы.

`-u` выводит только идентификатор пользователя.

Описание

Команда `id` выводит идентификатор пользователя, идентификатор группы и все группы для указанного пользователя. Если имя пользователя не указано, `id` выводит информацию о текущем пользователе.

passwd

Назначение

Изменяет пароль.

Синтаксис

```
passwd [имя_пользователя]
```

Опции

Нет

Описание

Команда `passwd` изменяет ваш пароль. Она запрашивает старый пароль, а затем новый пароль. Если вы являетесь привилегированным пользователем вы можете изменить пароль другого пользователя, указав его имя в качестве аргумента команды `passwd`.

whoami

Назначение

Выводит действительный идентификатор пользователя.

Синтаксис

`whoami [опции]`

Опции

`-help` выдает справку о команде.

`-version` выводит на стандартное устройство вывода информацию о версии программы.

Описание

Команда `whoami` выводит имя пользователя, ассоциированное с текущим действительным идентификатором пользователя (UID). Данная команда эквивалентна команде `id -un`.

12.9 Управление системой

df

Назначение

Выводит количество свободного и занятого пространства во всех смонтированных файловых системах.

Синтаксис

`df [опции] [файловая_система]`

Опции

`-a` выводит информацию обо всех файловых системах.

`-i` выводит информацию об индексных дескрипторах (inode) в случае соответствующей

организации диска.

`-T` выводит тип файловой системы.

`-t` тип выводит информацию только об указанных типах файловых систем.

`-x` тип исключает указанные типы файловых систем из выходных данных.

`–help` выводит справочное сообщение.

Описание

Команда `df` показывает количество свободного и занятого пространства в указанной файловой системе. Если вы хотите узнать, насколько заполнены все ваши диски, наберите команду `df` без аргументов. В этом случае команда `df` выводит информацию об использованной и доступной памяти всех смонтированных на данный момент файловых систем.

du

Назначение

Отображает размер дискового пространства, занятого файлами или каталогами.

Синтаксис

```
du [опции] [каталоги_или_файлы]
```

Опции

`–a` выводит информацию об использовании памяти для всех файлов (а не только для каталогов).

`–b` выводит информацию в байтах (а не в килобайтах).

`–c` выводит итоговую информацию об использовании дисковой памяти.

`–k` выводит информацию в килобайтах (по умолчанию).

`–s` выводит итоговую информацию об использовании дискового пространства без информации о каталогах.

Описание

Команда `du` отображает объем дисковой памяти (в килобайтах), занятой указанными файлами или каталогами. По умолчанию `du` отображает объем дисковой памяти, используемой каждым каталогом и подкаталогом. Обычно команда `du` применяется для вывода общего объема дисковой памяти занимаемой текущим каталогом. Например, вот так можно узнать детали использования дискового пространства каталогом `/var/log`:

```
du /var/log
```

free

Назначение

Выводит количество свободной и занятой памяти в системе.

Синтаксис

```
free [опции]
```

Опции

`–b` выводит объем памяти в байтах.

`-k` выводит объем памяти в килобайтах (по умолчанию).

`-m` выводит объем памяти в мегабайтах.

`-s n` повторяет команду каждые *n* секунд.

`-t` выводит строку с общим количеством свободной и используемой памяти.

Описание

Команда `free` выводит информацию о физической памяти (RAM) и области свопинга (на диске). В выходной информации отображается общее количество памяти, а также количество используемой и свободной памяти.

uname

Назначение

Выводит системную информацию, такую как тип машины и операционной системы.

Синтаксис

```
uname [опции]
```

Опции

`-a` выводит всю информацию.

`-m` выводит тип оборудования (например, `i586`).

`-n` выводит имя хоста машины.

`-p` выводит тип процессора (обычно `unknown`).

`-r` выводит номер выпуска операционной системы

`-s` выводит имя операционной системы.

`-v` выводит версию операционной системы (дата компиляции).

Описание

Команда `uname` выводит различную информацию о машине и операционной сстеме (Linux).

uptime

Назначение

Отображает время непрерывной работы системы.

Синтаксис

```
uptime
```

Опции

Нет

Описание

Утилита `uptime` показывает текущее время, время непрерывной работы системы, число пользователей в системе и среднюю загруженность системы за последние 1, 5, и 15 минут.

12.10 Работа с датой и временем

cal

Назначение

Просмотр календаря за любой месяц любого года.

Синтаксис

```
cal [-jy] [[номер_месяца] год]
```

Опции

`-j` выводит юлианские даты (дни с номерами от 1 до 366).

`-y` выводит календарь для всех месяцев текущего года.

Описание

Если набрать `cal` без опций, выведется календарь на текущий месяц. Если ввести `cal`, за которым следует число, то `cal` считает это число номером года и выводит календарь для этого года. Для вывода календаря за конкретный месяц конкретного года укажите номер месяца (1 – январь, 2 – февраль и так далее), а за ним номер года.

date

Назначение

Отображает текущую дату и время или устанавливает новую дату и время.

Синтаксис

```
date [опции] [+формат]
date [-su] [ММДДЧЧММ[[ВВ]ГГ][.СС]]
```

Опции

`-u` выводит или устанавливает время по Гринвичу (GMT).

Описание

Команда `date` без аргументов выводит текущую дату и время. С помощью аргумента `+format` можно указать формат отображения даты и времени. Чтобы получить полный список спецификаций формата, наберите: `man date`

Для установки даты введите команду `date` с датой и временем в формате `ММДДЧЧММ`, где каждая буква обозначает цифру (`ММ` – месяц, `ДД` – день, `ЧЧ` – часы и `ММ` – минуты). Вы также можете указать необязательные год (`ГГ`) и век (`ВВ`).

time

Назначение

Измеряет время выполнения команды

Синтаксис

```
time команда [аргументы]
```

Опции

Нет

Описание

Выполняет команду с заданными аргументами, и после ее завершения выдает следующую информацию:

`real` астрономическое время выполнения команды (между ее запуском и завершением).

`user` user CPU time, затраченное на выполнение команды (сумма значений `tms_utime` и `tms_cutime` в `struct tms`).

`sys` system CPU time (сумма значений `tms_stime` и `tms_cstime` в `struct tms`).

Время сообщается в секундах; если время больше минуты, результат имеет вид`минуты:секунды`.

СОДЕРЖАНИЕ