Операционная система «ОСь»
Руководство администратора
Листов 312

Москва
2017

АННОТАЦИЯ

Настоящий документ является руководством администратора операционной системы «ОСь» (далее — «ОСь» или ОС).

В документе рассмотрены общие вопросы администрирования, управления процессами, устройствами, файловой системой и печатью, использования командной оболочки bash, мониторинга системы и сбора статистики, резервного копирования и восстановления, планирования задач и настройки сети.

Документ предназначен для системного администратора ОС.

СОДЕРЖАНИЕ

1. ОБЩИЕ ПОЛОЖЕНИЯ

В данном документе рассмотрены общие вопросы администрирования, управления процессами, устройствами, файловой системой и печатью, программирования на языке «Bash», мониторинга системы и сбора статистики, резервного копирования и восстановления, планирования задач и настройки сети.

Работа с программами обычно сопровождается листингами кода.

Листинги всех команд, вводимых через терминал, выделяются моноширинным шрифтом:

```
$ ls
```

Обратите внимание на символ доллара перед командой. Символ доллара означает, что эту команду следует ввести от имени пользователя, а символ решётки означает, что команду нужно вводить от имени системного администратора (`root`):

```
# passwd user
```

Внутри угловых скобок (<>) указываются параметры, которые пользователю следует изменить или ввести:

```
# passwd <имя пользователя>
```

Если текст в угловых скобках содержит многоточие, это означает, что можно указать один и более объектов:

```
# ls <файл...>
```

Если текст заключён в квадратные скобки, это означает, что можно указывать любое количество объектов одного типа или не указывать ни одного:

```
# cat [<файл...>]
```

Текстовый комментарий в листинге отделяется двумя наклонными чертами (`//`):

```
# passwd <имя пользователя>  // изменить пароль пользователя
```

Для обозначения необязательных параметров команды используются квадратные скобки. В данном примере параметр `-r` является необязательным, а имя файла — обязательно:

```
# cp [-r] <имя файла>
```

Если в квадратных скобках два параметра, то последующий параметр обязателен при использовании предыдущего. В данном примере имя файла должно вводиться только в том случае, если перед именем задан параметр `-f`, но при этом параметр `-f` использовать не обязательно.

```
$ man [-f <имя файла>]
```

Имя файла может использоваться только с параметром `-f`, но не является обязательным, параметр `-f` также необязателен:

```
$ man [-f[ <имя файла>]]
```

Для разделения списка возможных вариантов используется прямая черта. Данный пример означает, что команда может быть выполнена в двух вариантах — `ifconfig <интерфейс> up` или `ifconfig <интерфейс> down`:

```
$ ifconfig <интерфейс> up|down
```

Элементы управления на клавиатуре называются «Клавишами» и выделяются следующим образом:

Клавиша <Enter>; сочетание клавиш <Ctrl+Alt+F2>.

Сообщения, выводимые на экран компьютера, выделяются кавычками, например:

«Установка завершена».

2. УСТАНОВКА И НАСТРОЙКА ОС

В данном разделе представлены различные варианты установки и настройки операционной системы.

2.1. Загрузка программы установки системы с установочного носителя

Для загрузки программы установки, выполните следующие действия:

– включите компьютер;

– настройте в BIOS загрузку системы с DVD-привода;

– вставьте в DVD-привод диск «Операционная система «ОСь» Текст программы. Загрузочный модуль»;

– перезагрузите компьютер.

П р и м е ч а н и е. Подробнее настройки BIOS описываются в паспорте и руководствах, поставляемых в составе аппаратного обеспечения.

В процессе запуска компьютера загрузится программа установки, появится окно с меню установки.

Внизу окна будет отображена строка обратного отсчета (длительностью 60 секунд), отсчитываемого до начала автоматической загрузки с выбранным пунктом меню по умолчанию «Установка ОС в графическом режиме».

Чтобы прекратить обратный отсчет, необходимо клавишами навигации (стрелки вверх или вниз) выбрать любой из вариантов, предлагаемых программой установки, при этом строка обратного отсчета перестанет отображаться и система перейдет в состояние ожидания действий администратора. Выберите один из пунктов меню и нажмите клавишу [Enter].

2.2. Установка ОС в графическом режиме

Для установки ОС в графическом режиме выберите пункт меню программы установки «Установить ОС в графическом режиме», нажмите клавишу [Enter] будет загружена программа установки ОС (необходимо подождать несколько минут). После загрузки программа установки проверит целостность носителя и отобразит графический интерфейс.

2.2.1. Настройка установки

Интерактивный графический диалог с администратором позволяет детально настроить процесс установки ОС. На экран выводятся пункты меню, которые можно настроить перед установкой. Пункты меню, которые ожидают действий администратора, помечены оранжевой иконкой с восклицательным знаком. Остальные пункты меню уже настроены по умолчанию, но их параметры могут быть изменены.

Для изменения параметров любого из пунктов меню, выберите его и следуйте указаниям мастера установки. После настройки нажмите на кнопку «Готово». Повторите эти действия для всех параметров, которые вы хотите изменить.

В правом верхнем углу окна программы установки отображается иконка раскладки клавиатуры. Её можно использовать для изменения языка ввода, так же для изменения языка ввода можно использовать клавишу [Caps Lock].

2.2.2. Выбор места для установки

Обязательный шаг, который должен выполнить администратор для установки в графическом режиме — выбрать жёсткий диск и разделы, на которые будет устанавливаться ОС. Для этого в окне «Обзор установки», необходимо щёлкнуть правой кнопкой мыши по пункту меню «Место установки» (рисунок 1).

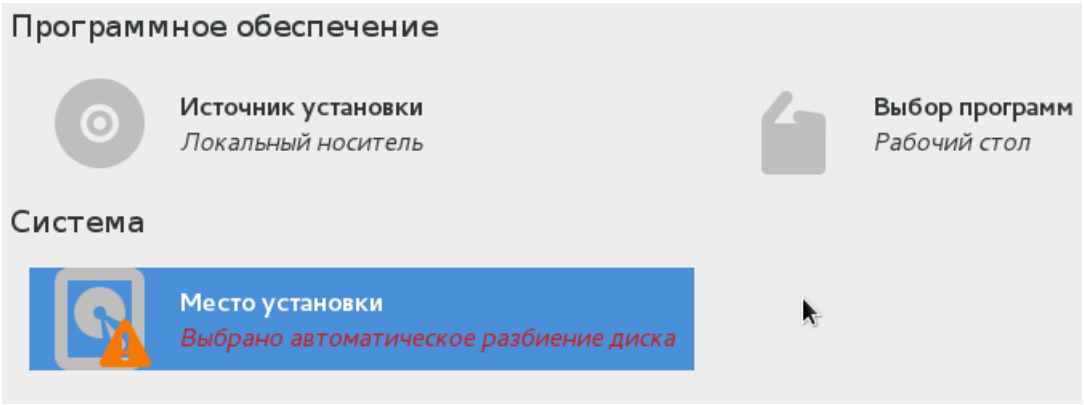

Рисунок 1 – Выбор места установки ОС

Отобразится следующее окно «Место установки».

Если установка ОС происходит на компьютере, не имеющим никакой предустановленной ОС, можно оставить параметры по умолчанию и нажать кнопку «Готово», программа установки автоматически произведёт разбиение жёсткого диска перед установкой ОС.

Если на жёстком диске находится ранее установленная ОС, есть возможность настроить разбиение разделов жёсткого диска вручную. Для этого в разделе «Другие параметры хранения данных», выберите пункт «Я настрою разделы» и нажмите кнопку «Готово». В следующем окне выбрать пункт «Неизвестный Linux» и щёлкая по кнопке «–» последовательно удалите разделы. После удаления разделов щёлкните правой кнопкой мыши последовательно по: пункту «Нажать здесь для их автоматического создания»; кнопке «Готово»; кнопке «Принять изменения».

Администратор может переключить программу установки в режим командной строки (клавиши [Ctrl+Alt+F2]) и самостоятельно настроить разбиение дисков, используя интерактивную программу `fdisk`.

2.2.3. Установка

Для начала установки в окне «Обзор установки» щёлкните правой кнопкой мыши по кнопке «Начать установку». В окне «Конфигурация» выберите пункт меню «Пароль root» и в отобразившемся окне «Пароль root» введите сложный пароль администратора и щёлкните по кнопке «Готово». В окне «Конфигурация» щёлкните по кнопке «Завершить настройку».

По окончании установки нажмите на кнопку «Перезагрузка». Операционная система «ОСь» установлена.

2.3. Установка ОС с базовым видеодрайвером

Режим интерактивного графического диалога с администратором. Используется базовый видеодрайвер, позволяющий установить ОС на компьютер с неподдерживаемым видеадаптером.

Для установки, выберите пункт меню программы установки «Установить ОС с базовым видеодрайвером», нажмите клавишу [Enter] будет загружена программа установки ОС (необходимо подождать несколько минут). После загрузки программа установки проверит целостность носителя и отобразит графический интерфейс. Процесс интерактивного диалога программы и администратора аналогичен диалогу режима «Установить ОС в графическом режиме».

2.4. Установка ОС в режиме командной строки

Режим интерактивного текстового диалога с администратором. Установка операционной системы в режиме командной строки может использоваться для установки ОС на компьютер в составе которого есть видеокарта, не поддерживаемая ОС. Так же, в случае нехватки оперативной памяти.

П р и м е ч а н и е. Установка ОС в режиме командной строки устанавливает ОС без графического интерфейса.

Для установки ОС в режиме командной строки выберите пункт меню программы установки «Установка ОС в режиме командной строки», нажмите клавишу [Enter]. Запустится программа установки ОС (необходимо подождать несколько минут). После загрузки программа проверит целостность носителя и выведет текстовой диалог.

Настройка установки построена в виде интерактивного диалога с администратором: администратор вводит команду и нажимает клавишу [Enter], система выполняет выбранное действие. Вывод информации происходит последовательно, просмотреть историю диалога можно используя комбинации клавиш [Shift+PageUp] и [Shift+PageDown].

2.4.1. Настройки установки

На экране отобразится окно программы «Установка». В псевдографическом интерфейсе отображаются пронумерованные пункты меню со значком [x] или [!]. Значок [!] означает, что нужно выбрать этот пункт меню и произвести настройку. Значок [x] означает, что настройка данного пункта меню завершена, но в неё могут быть внесены изменения.

Для того чтобы выбрать пункт меню, введите цифровое обозначение пункта меню и нажмите клавишу [Enter].

2.4.2. Настройка временных зон

Администратор может изменить настроенный часовой пояс. По умолчанию используется часовой пояс Москва. Для изменения часового пояса выберите данный пункт меню и следуйте указаниям программы установки.

2.4.3. Место установки

Необходимо выбрать место установки ОС (жёсткий диск) и разбиение разделов жёсткого диска. Программа установки предлагает возможность замены существующей системы, использования всего дискового пространства или использования только свободного пространства на диске. Для выбора места установки выберите данный пункт меню и следуйте указаниям программы установки.

Администратор может переключить программу установки в режим командной строки (клавиши [Ctrl+Alt+F2]) и самостоятельно настроить разбиение дисков, используя интерактивную программу `fdisk`.

2.4.4. Установка пароля администратора

В пункте «Пароль root» нужно обязательно задать сложный пароль администратора. Для создания или изменения пароля выберите пункт меню (введите цифровое обозначение пункта меню и нажмите клавишу [Enter]) и следуйте указаниям программы установки.

2.4.5. Создание пользователя

Для создания пользователя необходимо заполнить подпункты меню:

– «Ctreate user» — создать пользователя;

– «Fullname» — полное имя пользователя;

– «Username» — имя пользователя;

– «Пароль» — набрать пароль пользователя.

Необязательные подпункты меню:

– «Use password» — использовать пароль (если пароль не установлен, при первом входе система предложит установить пароль);

– «Administrator» — пользователь с правами администратора;

– «Groups»— указать группы пользователя.

2.4.6. Начало установки

Для начала установки нажмите клавиши [b] и [Enter]. На экране будет выводится прогресс установки пакетов. По окончании процесса установки программа установки предложит нажать клавишу [Enter] для завершения установки.

2.5. Установка ОС в автоматическом режиме

В данном программа автоматически установит ОС на компьютер. Дополнительных действий от администратора не потребуется.

ВНИМАНИЕ! УСТАНОВКА В АВТОМАТИЧЕСКОМ РЕЖИМЕ **УДАЛИТ ВСЕ ДАННЫЕ** С ЖЁСТКОГО ДИСКА.

Перед автоматической установкой убедитесь, что:

– на компьютере нет никаких важных данных;

– от компьютера отсоединены все внешние USB-накопители.

Для автоматической установки ОС выберите пункт меню программы установки «Установка ОС в автоматическом режиме», нажмите клавишу [Enter] будет загружена программа установки ОС (необходимо подождать несколько минут). После завершения установки система перезагрузится. Пароли загрузчика, пользователей root и user будут автоматически установлены в значение 12345678, которые необходимо изменить сразу после установки.

После установки ОС необходимо произвести первичную настройку системы.

2.6. Загрузка с жёсткого диска

Режим «Загрузка с жёсткого диска» позволяет отказаться от установки ОС и передаёт управление загрузчику ОС установленной на жёсткий диск.

2.7. Другие варианты установки (полная установка ОС)

Пункт меню «Другие варианты установки», содержит пункт «Полная установка ОС в автоматическом режиме».

В данном режиме программа автоматически установит все пакеты с диска «Операционная система «ОСь» на компьютер. Установка ОС будет произведена без участия администратора.

ВНИМАНИЕ! УСТАНОВКА В АВТОМАТИЧЕСКОМ РЕЖИМЕ **УДАЛИТ ВСЕ ДАННЫЕ** С ЖЁСТКОГО ДИСКА.

Перед автоматической установкой убедитесь, что:

– на компьютере нет никаких важных данных;

– от компьютера отсоединены все внешние USB-накопители.

Для автоматической полной установки ОС выберите пункт меню программы установки «Полная установка ОС в автоматическом режиме», нажмите кла-

вишу [Enter] будет загружена программа установки ОС (необходимо подождать несколько минут). После завершения установки система перезагрузится. Пароли загрузчика, пользователей root и user будут автоматически установлены в значение 12345678, которые необходимо изменить сразу после установки.

После установки ОС необходимо произвести первичную настройку системы.

2.8. Решение проблем

Пункт меню программы «Решение проблем» содержит пункты:

– «Восстановление установленной системы»;

– «Запустить проверку памяти».

2.8.1. Восстановление установленной системы

Пункт меню «Восстановление установленной системы» позволяет получить доступ к данным, конфигурационным и другим файлам, находящимся на жёстком диске. Если ОС перестала загружаться, выполните восстановление ОС вручную, используя данный режим.

2.8.2. Запуск проверки памяти

Пункт меню «Запустить проверку памяти» запускает программу проверки работы оперативной памяти компьютера. Проблемы в работе ОС могут быть вызваны сбоями в работе оперативной памяти.

3. ЗАГРУЗОЧНОЕ МЕНЮ

Старт операционной системы начинается с загрузочного меню. При помощи загрузочного меню, можно выбрать загрузку необходимой ОС (если установлено несколько ОС), можно передать ядру различные параметры до начала загрузки операционной системы.

Где:

– «ОС ОСь *версия ядра*» — обычная загрузка операционной системы;

– «Enter» — выбор варианта загрузки;

– «e» — изменение параметров ядра;

– «c» — переход в командную строку.

Из меню загрузчика можно продолжить загрузку операционной системы, нажав клавишу [Enter]. Если на машине установлено несколько операционных систем, загрузчик предлагает выбрать одну из них, используя стрелки клавиатуры.

3.1. Вход в меню загрузчика

Включите компьютер, дождитесь окончания тестов базовой системы ввода-вывода и появления окна загрузчика. По умолчанию загрузчик ждёт несколько секунд, после чего начинает загрузку операционной системы. Если во время приветствия загрузчика нажать клавишу [e], откроется меню редактирования параметров загрузки.

Меню загрузчика заблокировано до тех пор, пока не будет введён пароль загрузчика. Пароль загрузчика устанавливается при установке операционной системы, для того, чтобы продолжить работу, нажмите клавишу [p] и введите пароль.

3.2. Изменение параметров ядра

Для того чтобы изменить параметры ядра, нажмите клавишу [e] в меню загрузчика. Откроется редактор параметров ядра, использующийся для настройки параметров ядра до загрузки ОС.

Параметры ядра — это командная строка, в которой нужно задавать аргументы вида:

```
<параметр>=<значение>
```

Аргументы необходимо разделять пробелом. В строке загрузки ядра уже установлены некоторые аргументы, изменять их или удалять не рекомендуется.

Некоторые важные параметры ядра, которые можно установить при загрузке:

`1` загрузка системы в однопользовательском режиме. Пользователь сразу оказывается в режиме командной строки под учётной записью системного администратора. Применяется в случае возникновения серьёзных проблем: сбой системы, отказ жёстких дисков.

`single` аналогично предыдущему.

`init=<путь до программы>` вместо процесса `init` запустить указанный. Обычно передают `init=/bin/bash`, чтобы получить доступ к командной оболочке, минуя все этапы загрузки операционной системы. Используется, когда систему невозможно загрузить даже в однопользовательском режиме (проблема с `init`).

`apci=off` полностью отключает ACPI (Advanced Configuration and Power Interface). Данный параметр полезен на некоторых устройствах, когда после установки операционная система не загружается из-за проблем с оборудованием. Для того, чтобы принудительно включить ACPI, нужно передать `acpi=on`.

`debug` запуск в режиме отладки с выводом всех сообщений ядра на терминал.

3.3. Командная строка

Командная строка загрузчика поддерживает ограниченное количество bash-подобных команд. По клавише [Tab] выводится список команд дополняющих набранную в строке команду. Для выхода из режима командной строки нажмите клавишу [ESC].

4. РАБОТА В ОС

В разделе описываются наиболее важные аспекты работы системного администратора. Администрирование операционной системы происходит из терминала с использованием командной строки.

4.1. Вход системного администратора

Администрирование операционной системы должно производиться от имени системного администратора. Системный администратор в ОС всегда имеет специальное имя пользователя: `root`. Пароль системного администратора запрашивается при установке операционной системы.

При загрузке операционной системы появляется приглашение входа в систему. Чтобы войти в систему как системный администратор, введите в качестве имени пользователя `root`, а в качестве пароля — тот пароль, что был указан при установке системы.

Во время работы под учётной записью обычного пользователя можно также осуществить вход от системного администратора при помощи команды `su`. Использовать команду `su` можно только на этапе первоначальной настройки операционной системы.

4.1.1. Администрирование при помощи sudo

На этапе первоначальной настройки системный администратор имеет возможность назначить своему пользователю права системного администратора через команду `sudo`, добавив его в файл `sudoers`. Данная процедура позволит упростить администрирование при работе под учётной записью обычного пользователя.

П р и м е ч а н и е. КСЗ операционной системы ограничивает выполнение команд `su` и `sudo`.

Для предоставления данной возможности пользователю `user1`, нужно от имени системного администратора добавить в файл `/etc/sudoers` следующую строку:

```
user1 ALL=(ALL) ALL
```

Теперь для выполнения команд от имени системного администратора достаточно будет выполнить от имени пользователя `user1`:

```
$ sudo <команда с аргументами>
```

`sudo` требует ввод пользовательского пароля (а не пароля системного администратора). Чтобы организовать администрирование без необходимости ввода пароля, нужно изменить строку в файле `/etc/sudoers`:

```
user1 ALL=(ALL) NOPASSWD: ALL
```

Для изменения файла `/etc/sudoers` используйте команду `visudo`.

П р и м е ч а н и е. Данную команду можно выполнять только на этапе настройки, в процессе эксплуатации системы системные администраторы не должны использовать данную возможность и восстановить конфигурацию по умолчанию, если она была изменена.

Для возможности повышения полномочий, пользователя `user1` необходимо добавить в группу `wheel` выполнив команду:

```
usermod -aG 'wheel' user1
```

Так же для возможности повышения полномочий пользователя, можно при установке системы выбрать пункт «Сделать пользователя администратором».

4.2. Командная строка

Основное средство администрирования ОС — командная строка. Программы с графическим интерфейсом помогают администратору быстро выполнять типовые задачи по настройки системы, однако наиболее точная настройка достигается при работе с утилитами командной строки, которые предоставляют расширенные возможности администрирования.

Каждая команда, передаваемая пользователем системе, — инструкция, которую необходимо выполнить. Пока не нажата клавиша [Enter], строку команды можно редактировать, используя стандартные клавиши ввода на клавиатуре.

Интерфейс командной строки берёт своё начало от операционной системы UNIX, основные принципы которой сохранились до сих пор. Умение работать с командной строкой UNIX позволит работать в командной строке ОС. Умение системного администратора работать с командной строкой значительно упрощает процедуру администрирования и ускоряет её.

Команды интерпретируются и выполняются специальной программой — командной оболочкой. По умолчанию в ОС используется командная оболочка Bash. В файле `/etc/shells` находится список всех командных оболочек в системе:

```
# cat /etc/shells
/bin/sh
```

```
/bin/Bash
/sbin/nologin
/bin/tcsh
/bin/csh
```

Все дальнейшие примеры и указания касаются командной оболочки Bash.

Командная строка состоит из приглашения и вводимой команды. Приглашение — специальная последовательность символов, которая располагается в начале строки и задаёт начало области ввода команды, например:

```
user@home$
root@server1#
```

Приглашение командной оболочки пользователя оканчивается знаком доллара — «$», а приглашение для системного администратора всегда оканчивается символом решётки — «#». В дальнейшем, в примерах команд приглашение командной строки будет опускаться, а символ доллара или решётки — указывать от имени пользователя или администратора нужно выполнить ту или иную команду.

4.2.1. Доступ к командной строке из эмулятора терминала

Если работа происходит в графическом интерфейсе операционной системы, пользователь может использовать эмулятор терминала — специальное приложение, выполняющее все функции терминала. Для того, чтобы открыть окно эмулятора терминала, выберите следующий пункт меню: «Приложения — Системные — Терминал». Откроется окно с приглашением командной строки, работа в котором не отличается от работы с настоящим терминалом.

4.2.2. Виртуальные терминалы

Физический терминал компьютера в ОС — клавиатура и дисплей — поддерживают несколько виртуальных терминалов, которые могут работать как отдельные физические терминалы. В каждом виртуальном терминале выполняется отдельная сессия пользователя.

После загрузки операционной системы пользователь автоматически попадает в первый виртуальный терминал, в котором появляется приглашение ввода имени пользователя и пароля операционной системы. Если доступна графическая оболочка, на первом виртуальном терминале будет работать графическая сессия.

Для переключения между виртуальными терминалами используются сочетания клавиш от [Ctrl+Alt+F1] до [Ctrl+Alt+F7]. Переключение между виртуальными терминалами возможно в любой момент в процессе работы.

4.2.3. Состав команды

Каждая команда состоит из следующих далее частей.

4.2.3.1. Имя команды

Имя команды — её строковый идентификатор, совпадающий с именем программы. Некоторые команды (такие как `echo`) являются встроенными для командной оболочки и для их выполнения вызывается не программа, а внутренняя функция оболочки. Например, `/bin/echo` и `echo` — разные команды, первая является программой, вторая — функцией командной оболочки.

4.2.3.2. Аргумент

Практически любой команде можно передать аргументы командной строки. Аргументом может являться текстовая строка, соответствующая имени файла или другого объекта. Аргументы разделяются символом пробела. Чтобы передать в качестве аргумента строку символов с пробелами, нужно использовать экранирование, например символами кавычек (4.2.3.6):

```
$ echo ”Hello, world.”
Hello, world.
```

Подробно символы экранирования описаны в разделе .

Аргументы принято различать на непосредственно аргумент команды, обозначающий имя параметра данной команды и значение данного параметра. В большинстве программ с интерфейсом командной строки аргументы могут быть короткими и длинными. Короткие аргументы начинаются с одного символа «-» (дефис) и содержат один символ, следующий за дефисом (например, `-f`, `-o`, `-A`), а длинные аргументы начинаются с двух дефисов («–»), после которых следует ключевое слово аргумента (например, `--list`, `--force`, `--context`). Короткие аргументы можно объединять, например, набор аргументов `-l -s -a` будет эквивалентен аргументу `-lsa`. Формат аргументов командной строки носит рекомендательный характер и не все программы могут следовать общепринятому формату аргументов.

4.2.3.3. Формат команды

В тексте данного руководства «формат команды» обозначает совокупность имени команды и допустимых аргументов, а также порядка применения аргументов и параметры аргументов.

4.2.3.4. Перенаправления

Стандартный ввод и вывод каждой команды может быть перенаправлен в файл или другую программу. По умолчанию ввод и вывод команды связываются с пользовательским терминалом. Для указания источника ввода или назначения вывода (для обычной информации и для ошибок отдельно) используются специальные символы (<, >, | и >>). Поставив цифру перед знаком перенаправления, можно указать, какой поток вывода нужно перенаправить: 1 соответствует стандартному выводу (по умолчанию), 2 — поток ошибок `stderr`. Возможен особый режим перенаправления — перенаправление всех потоков, которой активируется символом `&` перед одним из знаков перенаправления <, > или >>. Также можно указать явно командному интерпретатору, какой вывод нужно перенаправить, в частности, для перенаправления потока ошибок в стандартный вывод нужно добавить к команде сочетание символов `2>&1`.

Как правило, порядок данных частей команды должен быть именно таким, однако некоторые из них (кроме имени команды) могут отсутствовать.

Далее приведены примеры.

Перенаправление всех ошибок в `/dev/null`:

```
$ ls 2>/dev/null
```

Перенаправление вывода команды в файл:

```
$ ls > out.txt
```

Перенаправление вывода команды в другую команду:

```
$ ls | grep file
```

Перенаправление всех потоков, включая поток ошибок в `/dev/null`:

```
$ ls non-exestent-file &> /dev/null
```

Перенаправление всех потоков, включая поток ошибок в другую команду:

```
$ ls non-exestent-file 2>&1 | cat
```

Перенаправление потока ошибок в файл `errors.log`:

```
find /etc -name passwd 2> errors.log
```

4.2.3.5. Универсальные параметры команд

Большинство команд не только подчиняются стандартизированному формату аргументов командной строки, но и имеют некоторые стандартные аргументы:

`-h, -?, --help` Выводит краткую справку о программе.

`-v, -V, --version` Выводит информацию о версии программы.

`-v, -vv, --verbose` «Подробный режим», программа выводит диагностическую информацию о выполняемых действиях, некоторые программы поддерживают разные уровни диагностических сообщений от менее подробных до более подробных путем указания числа (`-v1`, `-v2`) или параметра `-vv`, `-vvv`.

`-` Использование символа дефиса вместо аргумента файла часто означает, что файл нужно считать из стандартного ввода, что удобно про использовании команд в конвейере.

`--` Два символа дефиса обозначают окончание параметров, после этого любые аргументы не воспринимаются не как параметры.

Не все команды подчиняются данному правилу, поэтому иногда стоит попробовать несколько вариантов получения справки или информации о версии. Данные параметры опущены для большинства команд, если только они не имеют другое специальное значение.

Далее приведены примеры.

Показать краткую справку по команде `ls`:

```
$ ls --help
```

Команда `cat` в качестве входного файла использует стандартный ввод:

```
$ cat - < /etc/passwd
```

Команда `ls` отображает информацию по файлу с именем `-l`:

```
$ ls -- -l
```

При этом поведение нижеследующей команды отличается, так как `-l` является особым параметром для команды `ls`:

```
$ ls -l
```

4.2.3.6. Экранирование символов

Экранирование символов — замена управляющих символов на соответствующие текстовые подстановки. Символ экранирования «\» сообщает командной оболочке, что символ, стоящий за ним, надо воспринимать буквально. Символы, нуж-

дающиеся в экранировании: пробел « », кавычка (одинарная «'» и двойная «"»), обратный апостроф «`», доллар «$», и непосредственно сам символ экранирования «\».

При передаче аргумента в команду все управляющие символы командной оболочки необходимо экранировать одним из следующих способов:

– заключение аргумента в кавычки (одинарные или двойные);

– экранирование каждого спецсимвола.

При экранировании двойными кавычками командная оболочка воспринимает знак доллара и обратный апостроф как спецсимволы, поэтому внутри двойных кавычек необходимо экранировать знак доллара, обратный апостроф и саму двойную кавычку. Внутри одинарных кавычек нужно экранировать только одинарную кавычку.

Примеры.

```
$ echo "Hello world"
Hello world
$ echo 'Hello world'
Hello world
$ a=world
$ echo "Hello $a"
Hello world
$ echo 'Hello $a'
Hello $a
$ echo "Hello \$a"
Hello $a
$ echo "`pwd`"
/home/user
$ echo "\`pwd\`"
`pwd`
```

4.2.4. Значение, возвращаемое командами

Каждый процесс при завершении возвращает своему родительскому процессу специальный код завершения программы. Код может использоваться для получения результата выполнения программы и для проверки корректности её выполнения (возврата кода ошибки).

В ОС в случае успешного выполнения программа возвращает значение 0. Другие значения (все, отличные от 0) означают тот или иной код ошибки. Узнать код возврата программы можно с помощью специальной служебной переменной командной оболочки: (`$?`), например:

```
$ ls
...
$ echo $?
0
$ ls nonexistent
ls: невозможно получить доступ к unexistent: Нет такого
файла или каталога
$ echo $?
2
```

4.2.5. Объединение команд

Объединение нескольких команд в одной командной строке можно выполнить несколькими способами:

– «Последовательное выполнение». Команды выполняются одна за другой, не зависимо от результата их исполнения. В качестве разделителя выступает символ точки с запятой (`;`).

В качестве примера можно рассмотреть составную команду `du -sh; date`, которая выводит на экран содержимое домашнего каталога, а затем текущие дату и время. Это простой способ последовательного объединения команд, которые должны выполняться вместе или они используются для создания сложных псевдонимов;

– «Условное выполнение (И)». Проверяется код завершения первой программы, если он равен 0, производится выполнение второй программы и так далее. Таким образом, последняя команда выполнится только при успешном завершении всех предыдущих. В качестве разделителя выступают два символа амперсанд — `&&`. Пример:

```
$ touch file && echo OK
```

На экране появится слово «ОК», если файл был успешно создан;

– «Условное выполнение (ИЛИ)». Проверяется код завершения первой программы, если он не равен 0, производится выполнение второй программы и так

далее. Последняя команда будет запущена только если ни одна из предыдущих не закончилась успешно. В качестве разделителя выступают две вертикальные черты — `||`.

Пример:

```
$ touch file || echo Fail
```

Команда выведет на экран слово «Fail» в случае неудачи;

– «Конвейер». При выполнении программ в режиме конвейера их каналы ввода и вывода связываются так, что текстовые данные передаются через них последовательно, как по конвейеру. Этот механизм является одной из самых фундаментальных особенностей командной строки. В конвейер могут быть объединены последовательно несколько команд, возможно распараллеливание конвейера на несколько потоков при помощи программы `tee`.

Все программы могут быть запущены одновременно так, чтобы обеспечить интерактивность вывода при прохождении данных через цепочку программ. Если одна из программ в цепочке конвейера завершится (например, в случае ошибки), остальным будет отправлен специальный сигнал (`SIGPIPE`).

В качестве разделителя используется символ вертикальный черты (`|`).

Пример:

```
$ cat /etc/passwd | grep :/bin/sh | grep -oE ”^[a-z]+” |
sort
```

Данная команда выполняет чтение файла `/etc/passwd`, поиск в нём всех пользователей, у которых в качестве стандартной командной оболочки используется `sh`, фильтрует только имена этих пользователей и сортирует полученный список пользователей по алфавиту.

Командная оболочка Bash позволяет строить гибкие условные выражения (с использованием скобок и знака отрицания) из выполняющихся команд.

4.2.6. Задачи

Bash позволяет запускать задачи: пользовательские программы, которые работают в фоновом режиме. Для запуска задачи необходимо запустить программу, добавив в конце аргументов запуска символ «&».

Пример:

```
$ sleep 1 &
[1] 12443
```

Вывод команды отображает номер задачи в квадратных скобках, по которому к ней можно обратиться в дальнейшем и идентификатор процесса данной задачи.

Подробное описание команд управления задачами приведено в разделе 8.2.

4.2.7. Переменные окружения

Переменные окружения устанавливаются пользователем или сценариями командной оболочки. Начальный набор переменных окружения задается стартовыми сценариями операционной системы и сценариями, запускаемыми при регистрации пользователя в системе, например, `/etc/profile` и `/.Bash_profile`. Для имён переменных окружения принято использовать только прописные символы и знак подчёркивания.

Среда в ОС хранит настройки переменных окружения системы и отдельных программ. Каждый процесс при запуске получает свою копию переменных окружения.

Переменные окружения представляют собой набор пар «имя переменной» и «значение переменной». Переменная окружения задается вводом в командной строке пары `ИМЯ=”значение”`:

```
$ <ИМЯ>=”<значение>”
```

Например, команда:

```
$ PATH=”/usr/bin:/bin”
```

задаёт переменной `PATH` значение `/usr/bin:/bin`.

Формат команды для просмотра значения переменной окружения:

```
$ echo $<ИМЯ>
```

Например,

```
$ echo $HOME
/home/user
```

Просмотр списка переменных окружения, доступных в командной оболочке:

```
$ env
```

4.2.7.1. Стандартные переменные окружения

При старте командной оболочки, инициализируются переменные окружения, которые можно посмотреть с помощью команды `env`, например:

– `DISPLAY` — переменная используется графической подсистемой X11 и указывает на адрес X-сервера и номер используемого экрана;

– EDITOR — в этой переменной можно указать путь до программы-редактора текстовых файлов, которые будут вызываться запущенной программой;

– HOME — переменная содержит имя домашнего каталога текущего пользователя;

– PATH — Переменная окружения, содержащая список каталогов, разделённый символом (:). Этот список просматривается при каждом запуске команды — в нём производится поиск исполненных файлов с соответствующим команде именем. Примером значения переменной окружения может быть /bin:/usr/bin:/usr/local/bin. Как правило, значение этой переменной различается для простого пользователя и администратора, чтобы разделить их рабочий инструментарий;

– SHELL — имя текущей программы оболочки;

– TERM — тип терминала, используемого в настоящий момент. Эта переменная анализируется программами для того, чтобы варьировать свой интерфейс в зависимости от возможностей терминала;

– USER — имя текущего пользователя;

– (_) — символ подчёркивания содержит имя последней команды, выполненной в командной оболочке.

4.2.7.2. Установка переменной окружения

Установить переменную окружения процесса можно путём его выполнения в командной строке, указав перед именем программы пару ключ=значение. Если нужно установить переменную окружения для текущего сеанса командной оболочки, используется команда export.

Пример запуска программы с заданной переменной окружения:

```
$ LD_LIBRARY_PATH=`pwd` ./program
```

LD_LIBRARY_PATH устанавливается в значение текущего рабочего каталога, что дополнительно сообщает программе, о расположении библиотек.

Аналогичный пример, но переменная окружения задаётся для командной оболочке и передаётся всем запускаемым программам:

```
$ export LD_LIBRARY_PATH=`pwd`
$ ./program
```

4.2.7.3. Получение списка переменных окружения

Утилита `env` получает текущее окружение и модифицирует его.

Формат команды:

`env [<параметры>] [-] [<переменная>=<значение>]... [<команда> [<аргументы...>]]`

Если задано `<переменная>=<значение>` — модифицирует окружение для указанной команды, изменяя только те переменные окружения, которые указаны явно. При наличии параметра `-i` наследуемое окружение полностью игнорируется, и команда выполняется с тем окружением, которое определено командной строкой `env`.

Если команда не задана, то сформированное окружение выдается на стандартный вывод, по одной паре `имя=значение` в строке.

Параметры:

– `-i`, `--ignore-environment` — исполняет команду с пустым окружением;

– `-u`, `--unset=<переменная>` — убирает переменную из окружения.

Без параметров считается запущенной с ключом `-i`.

Аналогичная команда — `printenv`, выводит только переменные окружения.

Формат команды:

`printenv [<переменная...>]`

Если аргументы не указаны, выводит все переменные окружения, иначе выводит только указанные переменные.

4.2.8. Интерфейс командной оболочки

В ОС используется командная оболочка (или командный интерпретатор) Bash, который предоставляет пользователю интерфейс командной строки. Командная оболочка выполняет в том числе функции: редактирование команды, автодополнение, история команд.

4.2.8.1. Редактирование команд

При вводе команды пользователь перемещает курсор по вводимому тексту и изменяет текст.

Для перемещения курсора используются клавиши клавиатуры [влево] — [←] и [вправо] — [→]. Для перемещения курсора в начало вводимого текста необходимо

нажать [Home], для перемещения курсора в конец строки нажать [End]. Перемещение курсора влево на одно слово, нажать сочетание клавиш [Ctrl+влево]. Перемещение курсора на одно слово вправо, нажать сочетание клавиш [Ctrl+вправо].

Команда может располагаться на нескольких строках для удобства ввода длинных команд. Командная оболочка обычно самостоятельно выполняет перенос строк в длинных командах, однако чтобы выполнить объединение двух и более частей команды, в конце строки вставьте символ «\» и нажмите клавишу [Enter]. Например:

```
$ ls /etc /proc /etc/sysconfig /dev \
/usr/share/backgrounds
```

Если выполняется ввод многострочной команды или ввод данных в файл с помощью команды `cat`, для завершения ввода необходимо нажать сочетание клавиш [Ctrl+D].

Удаление символа перед курсором выполняется нажатием на клавишу [Backspace]. Удаление символа после курсора выполняется нажатием на клавишу [Del].

Для завершения процесса нажать сочетание клавиш [Ctrl+C]. Некоторые команды, например, `ping`, будут работать до тех пор, пока не будет нажато сочетание клавиш [Ctrl+C] или процесс не будет завершен принудительно другим способом, например, командой `kill/killall`.

Чтобы «усыпить» процесс, нужно нажать сочетание клавиш [Ctrl+Z], это не завершит процесс, но приостановит его. Приостановленный процесс нужно либо завершить, либо возобновить с помощью команды `kill/killall`. Найти процесс, который «спит», поможет команда `ps`.

4.2.8.2. История команд

Команды, набранные пользователем, оболочка Bash запоминает. Список команд, введенных пользователем, просматривается нажатием клавиш на клавиатуре [стрелка вверх] и [стрелка вниз]. Нажатием клавиши [стрелка вверх] список введенных команд «прокручивается» от последней к первой, а по нажатию клавиши [стрелка вниз] список введенных команд «прокручивается» от первой к последней. Введенная ранее в оболочку Bash команда отображается в командной строке. Пользователь выбирает, редактирует и вводит команду на исполнение оболочки Bash нажатием клавиши [Enter].

Список команд сохраняется между сеансами работы пользователя в момент завершения работы оболочки Bash. Накопленный за время работы список команд дописывается в конец файла `~/.bash_history`, находящийся в домашнем каталоге пользователя. При следующем запуске командной оболочки Bash происходит считывание файла `~/.bash_history`. Количество сохраняемых команд в `~/.bash_history` ограничено текущей настройкой, например 500 команд.

Удобный просмотр истории обеспечивает команда `history`.

Формат команды:

```
history [-c]
```

Без аргументов выводит на экран содержимое истории. Параметр `-c` полностью очищает историю команд.

4.2.8.3. Поиск по истории команд

Для того чтобы найти в истории ранее набранную команду, нужно нажать сочетание клавиш [Ctrl+R] и начать вводить строку поиска. Командная оболочка автоматически начнёт предлагать вариант команды, основываясь на истории ранее набранных команд. После того как на экране появилась нужная строчка, нужно отправить её на выполнение, нажав клавишу [Enter], или отредактировать её, нажав одну из клавиш перемещения курсора.

Возможен другой вид поиска по истории: при помощи автодополнения, который описан в разделе «Автодополнение» ниже.

4.2.8.4. Псевдоним

Пользователь использует псевдонимы команд для сокращения длины набираемой строки команды. Например, псевдоним — `grep`, а полная команда — `grep --color=auto`.

В конфигурационных файлах командной оболочки Bash определены сокращения, список которых выводится на экран после ввода команды `alias` без параметров:

```
$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias vi='vim'
```

```
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

Псевдонимы не наследуются с окружением.

Для назначения псевдонима пользователь выполняет команду `alias` с параметром, например:

```
$ alias ls='ls --color=auto'
```

Однако, псевдоним, заданный таким образом не сохранится после выхода из текущей командной оболочки. Обычно псевдонимы назначают через конфигурационный файл командной оболочки `~/.bashrc`. Данный файл доступен для редактирования каждым пользователем операционной системы. Чтобы добавить персональные сокращения команд, откройте файл `~/.bashrc` любым текстовым редактором, например, `vim` (описание работы с `vim` приведено в разделе 4.8), и добавьте в него несколько строк с командой `alias`:

```
alias t='tar'
alias s='secon'
alias n='newrole'
```

Пользователь может устанавливать сокращения по своему усмотрению. Для каждого пользователя будет свой собственный набор сокращений.

П р и м е ч а н и е. Так как пользователь может заменять существующие команды при помощи `alias`, при разработке сценариев на языке Bash необходимо учитывать и это и обращаться к командам не по имени, например, `grep`, `cat`, `du`, а по полному пути: `/bin/grep`, `/bin/cat`, `/usr/bin/du`.

Отменить действие синонима можно при помощи команды `unalias`:

```
$ unalias ls grep
```

Запуск с параметром `-a` удаляет все псевдонимы. Это иногда полезно для сценариев, чтобы убедиться, что команды действительно те, которые ожидает разработчик сценария, а не те, что установил пользователь.

```
$ unalias -a
```

4.2.8.5. Автодополнение

Автодополнение позволяет быстро набирать команды и пути в файловой системе. Если после набора неполного имени команды или файла нажать клавишу

[Tab], командная оболочка Bash предложит возможные варианты продолжения имени команды или файла по неполному имени команды или файла или дополнит его полностью, если вариант всего один, например:

```
$ ls /<TAB>
bin/            initrd.img.old  sbin/           usr/
boot/           lib/            var/
cdrom/          media/          srv/            vmlinuz
...
$ ls /
```

В операционной системе присутствует другой метод автодополнения команд, основанный на истории введённых ранее команд. Начните набирать команду и нажмите клавишу [PageUp] или [PageDown] — система будет подставлять команды из истории. Перемещение по истории команд происходит последовательного нажатия клавиш [PageUp] или [PageDown].

4.2.8.6. Шаблоны

Удобное средство для работы со многими файлами — использование шаблона. При наборе текста команды имена файлов нужно перечислять через пробел либо указать шаблон, по которому командная оболочка Bash подставит попадающие под маску файлы.

Шаблон состоит из символов, используемых в имени файла и/или из управляющих символов.

Возможны следующие управляющие символы:

– `*` — звездочка, означает любой символ, повторяющийся любое число раз;

– `?` — знак вопроса, означает точно один любой символ;

– `[]` — квадратные скобки, набор символов, заключенный в скобки, например `[abc]` соответствует одному из возможных перечисленных символов: `a`, `b` или `c`. Для обозначения диапазона символов в алфавитном порядке используется дефис: `[a-c]`, что эквивалентно `[abc]`;

– `{}` — фигурные скобки, позволяют перечислять несколько выражений подряд, например: `{foo,bar}*` будет соответствовать всем файлам, начинающимся с `foo` или `bar`. Фигурные скобки не являются шаблонами как таковыми — они выполняют роль генератора списков.

П р и м е ч а н и е. Шаблоны игнорируют файлы, начинающиеся с точки. Так, шаблон `*.txt` будет соответствовать `file.txt`, но не будет соответствовать `.hidden.txt`. Для охвата шаблонами файлов, начинающихся с точки, нужно явно указать точку в начале: `.*.txt` будет соответствовать файлу `.hidden.txt`, но не `file.txt`. Шаблон, соответствующий вообще всем файлам: `{,.}*.txt`.

Примеры шаблонов:

```
$ mkdir ~/test_dir
$ cd ~/test_dir
$ touch {1,2,3,11,file,.hidden}.txt foot  football  bard
$ ls -A
11.txt  1.txt  2.txt  3.txt  bard  file.txt  foot  football
.hidden.txt
$ ls *.txt
11.txt  1.txt  2.txt  3.txt  file.txt
$ ls ?.txt
1.txt  2.txt  3.txt
$ ls [12].txt
1.txt  2.txt
$ ls {foo,bar}*
foot  football  bard
$ ls *[a-z].txt
file.txt
$ ls .*[a-z].txt
.hidden.txt
$ ls {,.}*[a-z].txt
file.txt  .hidden.txt
```

Шаблоны в фигурных скобках можно использовать при создании файлов:

```
$ touch {foo,bar}{dea,dbe,ef}
$ ls
bardbe  bardea  baref  foodbe  foodea  fooef
```

При создании файлов использовать другие шаблоны (звёздочку или группы символов) нельзя. Это связано с тем, что `{}` является не стандартным шаблоном, а генератором комбинаций:

```
$ echo {0,1,2,3,4,5,6,7,8,9}
```

```
0 1 2 3 4 5 6 7 8 9
$ echo {0,1}{0,1}
00 01 10 11
```

Возможно использование фигурных скобок для комбинирования строк:

```
$ echo -e "\n"{"Яблоко","Груша"}"
"{"зелёного","красного","жёлтого"}" цвета"
Яблоко зелёного цвета
Яблоко красного цвета
Яблоко жёлтого цвета
Груша зелёного цвета
Груша красного цвета
Груша жёлтого цвета
```

Подобные комбинации полезны при установке нескольких программ:

```
# yum install lib{acl,curl,ev}-devel
```

Данная команда запросит установку пакетов `libacl-devel`, `libcurl-devel` и `libev-devel`.

4.2.8.7. Справочная система

В ОС существует встроенная справочная система по основным командам. Для получения справки по команде администратор вводит следующий текст в командной строке:

```
$ man <имя команды>
```

Откроется окно, содержащее справочное руководство по `<имя команды>`. Администратор перемещает текст справки в окне, нажимая на клавиатуре клавиши [стрелка вверх], [стрелка вниз], [Page Up], [Page Down] и [/].

Справочная система выводит на экран описание, список параметров, примеры команды и аналогичные команды.

Дополнительно к справочной системе некоторые программы выводят на экран краткие сведения о параметрах команды после ввода администратором `<имя команды>` с параметром `--help` или `-h`. Например:

```
$ cat --help
Usage: cat [OPTION]... [FILE]...
Concatenate FILE(s), or standard input, to standard output.
...
```

4.2.8.8. Очистка экрана

Для очистки экрана от результатов вывода предыдущих команд используется утилита `clear`.

Формат команды:

```
clear
```

В зависимости от типа используемого терминала команда автоматически выбирает метод очистки.

4.2.8.9. Сброс состояния терминала

Иногда полезно не только очистить терминал, но и сбросить все изменения. Для сброса терминала используется утилита `reset`.

Формат команды:

```
reset
```

Утилита `reset` полезна, когда в терминале по ошибке был открыт файл с двоичными данными, нарушивший формат ввода или вывода.

4.2.8.10. Выход из командной оболочки

Команда `exit` завершает текущий сеанс командной оболочки, в случае использования внутри сценария — завершает сценарий с указанным кодом возврата.

Формат команды:

```
exit [<код завершения>]
```

Если указан код завершения, сценарий завершится с указанным кодом.

4.2.8.11. Перечень сочетаний клавиш

Работа с командным интерпретатором значительно упрощается и ускоряется при использовании специальных сочетаний клавиш. В таблице 1 приведены основные сочетания клавиш командной оболочки.

Таблица 1 – Сочетания клавиш командной оболочки

Сочетание клавиш	Описание
Ctrl+C	Завершить текущую задачу (сигнал `SIGINT`)
Crtl+Z	Остановить текущую задачу

Сочетание клавиш	Описание
Ctrl+D	Завершить ввод или выйти из текущего сеанса
!!	Повторить предыдущую команду
Ctrl+A	Переход в начало строки
Ctrl+B	Переход на 1 символ назад
Ctrl+D	Удаляет символ под курсором (аналог [Delete])
Ctrl+E	Переход к концу строки
Ctrl+F	Переход на 1 символ вперёд
Ctrl+K	Удаляет всё, до конца строки
Ctrl+L	Очищает экран, аналог команды `clear`
Ctrl+R	Поиск по истории или повторение поиска (листание результатов поиска)
Ctrl+J	Прекращает поиск и позволяет отредактировать найденную команду
Ctrl+T	Меняет символ под курсором на предыдущий («тянет» предыдущий символ к концу строки)
Ctrl+U	Удаляет все символы слева от курсора до начала строки
Ctrl+W	Удаляет символы слева от курсора до начала слова.
Ctrl+XX	Переходит от текущей позиции курса в начало строки и обратно
Ctrl+X @	Показывает возможные дополнения имени хоста (имена берутся из `/etc/hosts`)
Alt+<	Переход к первой команде в истории команд
Alt+>	Переход к последней команде в истории
Alt+?	Показывает список возможных дополнений команды
Alt+*	Вставляет все возможные дополнений команды в строку команд
Alt+/	Пытается дополнить имя файла (аналогично табуляции)
Alt+.	Вставляет последний аргумент предыдущей команды
Alt+b	Сдвигает курсор влево на 1 слово
Alt+c	Делает букву под курсором большой, а остальные, до конца слова, маленькими.
Alt+d	Удаляет символы с текущей позиции курсора и до конца слова
Alt+f	Передвигает курсор на одно слово вперёд

Сочетание клавиш	Описание
Alt+l	Делает все буквы с текущей позиции курсора и до конца слова маленькими
Alt+t	Меняет местами слова под курсором и предыдущее
Alt+u	Переводит буквы с текущей позиции курсора и до конца слова в верхний регистр
Alt+BckSp	Удаляет символы с текущей позиции курсора до начала слова
TAB	Дополнение команды (иногда нужно нажать 2 раза)
*+TAB	Отображает все подкаталоги, включая скрытые
~+TAB	Выводит всех пользователей
$+TAB	Список всех доступных переменных окружения
@+TAB	Имена хостов из `/etc/hosts`
=+TAB	Аналог команды `ls` (содержимое текущего каталога)

4.2.9. Полезные приёмы при работе с Bash

Всегда используйте клавишу [TAB] при вводе имён команд и файлов:

– это минимизирует ошибки;

– если файл не существует, вы быстро об этом узнаете.

Автодополение по клавише [TAB] работает не только для имён файлов и команд, но и внутри некоторых команд, например, внутри команды `yum` — выводит список команд `yum`.

Быстрое переименование или создание резервной копии:

```
$ cp file{,.bak}
$ mv file{,.bak}
```

Аналогично:

```
$ cp file file.bak
$ mv file file.bak
```

Может быть полезно, если имя файла длинное.

Bash разделяет аргументы, используя специальную переменную IFS (Internal Field Separator). Эту переменную можно переопределить, чтобы выполнить некоторые необходимые команды максимально быстро, например:

```
$ IFS=":"
```

```
    $ read login pass uid gid realname homedir shell <
/etc/passwd
    $ echo $login,$pass,$uid,$gid,$realname,$homedir,$shell
    root,x,0,0,root,/root,/bin/bash
    $ unset IFS
```

Команда `unset` необходима, иначе Bash будет неправильно воспринимать все остальные команды.

4.3. Программирование на языке командного интерпретатора Bash

Командный язык Bash применяется администратором для управления компьютером с использованием интерпретатора команд `bash`.

Интерпретатор `bash` вызывается оболочкой Bash. Оболочка Bash считывает со стандартного устройства ввода команду и выполняет её.

Оболочка Bash обеспечивает взаимодействие администратора с операционной системой. С применением оболочки Bash администратор использует следующие функции по модификации команд:

– объединение команд для образования новых команд;

– передача позиционных параметров в команде;

– добавление или переименовывание команд;

– выполнение команды внутри циклов или по определенному условию;

– создание команды для локального выполнения без риска вступления в конфликт с командами других пользователей;

– выполнение команды в фоновом режиме;

– перенаправление ввода исходных данных для команды от одного источника к другому и перенаправление вывода данных в файл, на терминал, принтер или другой команде.

В языке программирования оболочки Bash используются:

– переменные;

– управляющие структуры типа `if`;

– подпрограммы, в том числе командные файлы;

– передача параметров;

– обработка прерываний.

Язык программирования поддерживает те же возможности, что и командная строка: допускается использование шаблонов, перенаправлений ввода и вывода, объединения команд.

Администратор использует оболочку Bash для:

– группировки команд, когда псевдонима недостаточно;

– автоматизации рутинных действий процесса администрирования или работы;

– написания своих программ-сценариев, выполняющих определенные задачи.

Некоторые программы в ОС являются Bash-сценариями, так как Bash подходит для решения множества задач.

4.3.1. Процедуры языка Bash

Процедурой языка Bash является командный файл. Существует два способа вызова администратором процедуры на выполнение:

```
$ sh program
```

Здесь `program` — некоторый командный файл. Сделать его исполняемым:

```
$ chmod 755 program
```

Запуск сценария на выполнение:

```
$ ./program
```

Для включения имени командного файла в перечень стандартных команд оболочки Bash администратор копирует командный файл в каталог `/usr/bin`.

Для указания операционной системе, что в качестве интерпретатора команд должна использоваться программа `sh`, в первой строке сценария пишут:

```
#!/bin/sh
```

или:

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

Во втором случае командным интерпретатором будет программа Bash. Возможно использование других программ-интерпретаторов: `sed`, `python`, `awk`.

4.3.2. Синтаксис

При разработке сценариев нужно выполнять правила работы с командной строкой:

– команды должны располагаться на разных строках, если нужно расположить две команды на одной строке — они должны быть объединены одним из операторов объединения (см. раздел 4.2.5);

– комментарий начинается с символа решетки: «#»;

– при присвоении значения переменной нельзя ставить пробел ни до, ни после знака равенства:

```
a=”foo”   # правильно
a = ”foo” # ошибка! Неправильный синтаксис
```

– строки, содержащие пробел, должны быть заключены в двойные или одинарные кавычки:

```
a=foo       # корректно - строка не содержит пробелов
a=foo bar   # ошибка!
a=”foo bar” # правильно
```

– в двойных кавычках («”») происходит интерпретация переменных, в одинарных кавычках («’») переменные будут выглядеть как текст;

– кавычка внутри кавычек экранируется знаком «\»;

– если нужно присвоить результат выполнения команды, а не саму команду, команда должна быть заключена либо в `“`, либо в `$()`;

– при подстановке переменных перед ними ставится знак доллара: `$var`, имя переменной отделяется фигурными скобками: `${var}`.

4.3.3. Переменные Bash

Все переменные в Bash — строки. Правила работа с переменными следующие:

– назначение значения переменной производится при помощи пары `имя=значение`;

– доступ к переменной — по имени со знаком `$` перед названием переменной:

```
fruit=apple #определение
echo $fruit #доступ
apple       #результат echo
```

– определение строки, содержащей пробел:

```
fruits=”apple, orange”
```

– объединение строк:

```
fruit=apple        # создаётся переменная
fruit=pine$fruit   # в начало переменной дописывается строка
```

```
echo $fruit        # вывод переменной на экран
pineapple
fruite=apple
wine=${fruite}jack
echo $wine
applejack
```

– в некоторых случаях, когда интерпретатор не может однозначно определить имя переменной, он считает границей переменной первый недопустимый символ, поэтому желательно заключать имя переменной в фигурные скобки:

```
dir=”/usr/bin”
#подразумевается dir, так как символ / недопустим
command=$dir/cat
dir=”/usr/bin/”
#команда некорректна, доступ к dircat, а не dir
command=$dircat
#выход из положения - использование фигурных скобок
command=${dir}car           #правильно
```

Переменная может быть:

– частью полного имени файла:

```
d=/usr/bin
echo $d/cat
```

– частью команды:

```
param=-l
ls $param   #то же самое что ls -l
```

– командой:

```
LS=/bin/ls
$LS -l      #то же самое что ls -la
```

– результатом выполнения другой команды

```
list=‘ls‘  # результат работы команды ls окажется внутри переменной list
#или
list=$(ls)
echo $list
```

Внутри команды, присваиваемой переменной, нельзя использовать символы «>», «<», «&» (символы, обозначающие перенаправление и фоновый режим).

Переменные окружения доступны внутри Bash-сценария, обращение к ним происходит как к обычным переменным.

В Bash имеются особые переменные, которые определяются самой оболочкой:

– `$?` — результат выполнения предыдущей команды;

– `$1` — первый аргумент командной строки;

– `$2` — второй аргумент командной строки (и т. д. до `$9`);

– `$@` — все аргументы командной строки, переданные сценарию;

– `$*` — все аргументы командной строки, переданные сценарию, объединённые в одну строку;

– `$#` — количество аргументов, переданных сценарию;

– `$$` — идентификатор процесса, под которым выполняется сценарий.

4.3.4. Операции над строками

Bash поддерживает некоторые операции над строками, которые не требуют использования внешних команд: поиск и замена, регулярные выражения, подсчёт количества символов.

Список возможных операций приведён в таблице 2. В качестве имени переменной используется `var`.

Таблица 2 – Операции над строками встроенными средствами Bash

Синтаксис операции	Описание
`${#var}`	Вычисление длины строки
`${var:N}`	Подстрока строки `var`, начиная с позиции `N` (отбрасывает первые `N` символов)
`${var:N:M}`	Подстрока строки `var`, начиная с позиции `N` длины `M`
`${var#<шаблон>}`	Удаляет наименьшее совпадение с `<шаблон>` от начала строки
`${var##<шаблон>}`	Удаляет наибольшее совпадение с `<шаблон>` от начала строки
`${var%<шаблон>}`	Удаляет наименьшее совпадение с `<шаблон>` с конца строки

Синтаксис операции	Описание
${var%%<шаблон>}	Удаляет наибольшее совпадение с <шаблон> с конца строки
${var/<шаблон>/<замена>}	Замена первого совпадения с <шаблон> на <замена>
${var//<шаблон>/<замена>}	Замена всех совпадений с <шаблон> на <замена>
${var/#<шаблон>/<замена>}	Замена совпадения с <шаблон> на <замена> от начала строки
${var/%<шаблон>/<замена>}	Замена совпадения с <шаблон> на <замена> с конца строки

Пример.

```
$ var=”This is a test string”
$ echo ${var}
This is a test string
$ echo ${#var}
21
$ echo ${var:3}
s is a test string
$ echo ${var:5:2}
is
$ echo ${var/is/si}
Thsi is a test string
$ echo ${var//is/si}
Thsi si a test string
$ echo ${var//?s/si}
Thsi si a tsitsitring
$ echo ${var/#??/00}
00is is a test string
$ echo ${var/%??/00}
This is a test stri00
$ echo ${var#* }
is a test string
$ echo ${var##* }
```

```
string
$ echo ${var##This}
is a test string
$ echo ${var%* }
This is a test string
$ echo ${var% *}
This is a test
$ echo ${var%% *}
This
```

Пример удаления символов или цифр из строки.

```
$ var=”ABCabc123”
$ echo ${var//[0-9]/}
ABCabc
$ echo ${var//[a-z]/}
123
```

Пример замены расширения файла.

```
$ file_name=image.png
$ echo ${file_name%.png}.jpg
image.jpg
$ echo ${file_name%.*}.jpg //заменит любое расширение
image.jpg
```

4.3.5. Условный оператор `if`

Синтаксис оператора `if`:

```
if <если эта команда выполняется успешно, то>;
then <выполнить все следующие команды до fi/else>;
[else <иначе выполнить следующие команды до fi>]
fi
```

Ключевые слова `if`, `then`, `else` и `fi` пишутся с начала строки, `else` является необязательным. Успешное выполнение команды означает, что она возвращает значение `0`. Любое другое возвращаемое значение означает ошибку. Возможны вложенные `if`. Для `else if` есть сокращение `elif`, которое одновременно сокращает `fi`.

Пример:

```
#!/bin/sh
if grep ^test: /etc/passwd
then echo ”Пользователь test существует”
else echo ”Пользователь test не зарегистрирован”
fi
```

Как правило, `if` используется совместно с командой `test`, которую опускают и используют квадратные скобки. В таком случае условие принимает следующий вид:

```
if [ <условие> ]
then
    ...
else
    ...
fi
```

Подробно условия команды `test` и примеры описаны в 4.3.6.

4.3.6. Команда `test`

Вычисление логических выражений, чаще всего используется в комбинации с условным оператором `if`. Команда, как правило, заменяется квадратными скобками: `[ ]` (скобки необходимо отделять от выражения пробелом).

Имеется три типа проверок:

– сравнение числовых значений;

– определение типа файла;

– сравнение строк.

Синтаксис команды сравнения чисел `N -op M`, где `N`, `M` — числа, `op` принимает значения:

– `eq` — равно;

– `ne` — не равно;

– `gt` — больше;

– `lt` — меньше;

– `ge` — больше или равно;

– `le` — меньше или равно.

Синтаксис команды определения типа файла `-op <имя файла>`, где `op` принимает значения:

– `e` — файл существует, при этом не важен тип данного файла;

– `s` — файл существует и не пуст (размер файла больше нуля);

– `f` — файл существует и является обычным файлом (см. 7.1.2);

– `d` — файл существует и является каталогом (см. 7.1.3);

– `c` — файл существует и является символьным устройством (см. 7.1.4);

– `b` — файл существует и является блочным устройством (см. 7.1.4);

– `p` — файл существует и является именованным каналом (см. 7.1.4);

– `h` — файл существует и является символьной ссылкой (см. 7.1.6);

– `L` — файл существует и является символьной ссылкой (см. 7.1.6);

– `S` — файл существует и является сокетом (см. 7.1.7);

– `r` — файл существует и доступен для чтения;

– `w` — файл существует и доступен для записи;

– `x` — файл существует и доступен на выполнение;

– `u` — файл существует и имеет установленный флаг SUID;

– `g` — файл существует и имеет установленный флаг SGID;

– `k` — файл существует и имеет установленный sticky bit;

– `G` — файл существует и принадлежит группе пользователя;

– `O` — файл существует и принадлежит пользователю;

– `N` — файл существует и был изменён с момента последнего чтения;

– `t` — указанный файловый дескриптор открыт в текущем терминале, при этом вместо имени файла необходимо указывать файловый дескриптор.

При выполнении операций над файлами все символические ссылки раскрываются.

Синтаксис команды сравнения файлов `<файл 1> -op <файл 2>`, где `<файл 1>` и `<файл 2>` — некоторые файлы, а `op` принимает одно из следующих значений:

– `ef` — файлы расположены на одном устройстве и имеют одинаковые индексные дескрипторы;

– `nt`typewriter`<файл 1>` новее, чем `<файл 2>`;

– `ot`typewriter`<файл 1>` старше, чем `<файл 2>`.

Синтаксис команды сравнения строк `<строка 1> op <строка 2>`, где `<строка 1>`, `<строка 2>` — строки, `op` принимает значения (обратите внимание на отсутствие дефиса перед операцией):

– = — эквивалентность;

– == — эквивалентность;

– != — не эквивалентность;

– > — истина, если <строка 2> идёт после <строка 1> в смысле сортировки;

– < — истина, если <строка 2> идёт до <строка 1> в смысле сортировки.

Синтаксис команды проверки строк имеет вид -op S, где S — строка, op принимает следующие значения:

– z — строка нулевой длины;

– n — строка не нулевой длины.

Если оператор не указан (задана только строка), это подразумевает проверку на строку не нулевой длины.

Проверки разных типов могут быть объединены логическими операциями, которые имеют синтаксис expr1 -op expr2, где expr1 и expr2 — некоторые выражения, а op может принимать одно из следующих значений:

– a — логическая операция «И»;

– o — логическая операция «ИЛИ».

Также существует операция логического отрицания, которая имеет формат ! expr, где expr — некоторое выражение. Выражения могут быть сгруппированы при помощи круглых скобок, при этом необходимо отделять круглые скобки при помощи пробела.

Пример:

```
# проверка, что $1 доступен на чтение, а $2 - на запись
if [ -w $2 -a -r $1 ]
then cat $1 >> $2
else echo ”cannot append”
fi
```

Иногда может использоваться без оператора if для краткости. Вместо условного оператора — объединение команд:

```
[ <условие> ] && <команда, если условие истинно>
[ <условие> ] || <команда, если условие ложно>
```

Пример:

```
$ [ ”1” = ”1” ] && echo ”Истина”
Истина
```

```
$ [ ”1” = ”2” ] && echo ”Истина” //ложь, команда не
выполнится
$ [ ”1” = ”2” ] || echo ”Ложь”
Ложь
$ [ ”1” = ”1” ] || echo ”Ложь” //истина, команда не
выполнится
```

Перечень примеров допустимых условий перечислен в таблице 3. Для наглядности скобки опущены.

Таблица 3 – Примеры логических операций Bash

Логическая операция	Описание
`! -f $HOME/.profile`	Истина, если отсутствует файл `.profile` в домашнем каталоге пользователя
`-s $MAIL`	Проверка наличии почты в почтовом ящике пользователя. Переменная окружения `$MAIL` содержит путь до файла с почтовыми сообщениями пользователя, и если он не пуст, выражение будет истинным
`$a = ”N” -o $a = ”n”`	Переменная `$a` содержит ответ пользователя на запрос о продолжении выполнения сценария. Выражение будет истинным, если ответ пользователя совпадает с «N» или «n»
`$n -ge 10 -a $n -le 20`	Истина, если число в переменной `$n` лежит в диапазоне от 10 до 20 включительно
`$# -ne 0`	Количество аргументов не равно нулю
`$STR = ”test”`	Истина, если строка `$STR` эквивалентна «test»

4.3.7. Оператор цикла `for`

Пусть имеется сценарий `makelist`:

```
$ cat makelist
sort +1 -2 people | tr -d -9 | pr -h Distribution | lpr
```

Если вместо одного файла `people` имеется несколько, например:

`adminpeople, hardpeople, softpeople,...,`

то необходимо повторить выполнение процедуры с различными файлами. Это выполняется с помощью оператора `for`.

Синтаксис оператора `for`:

```
for <переменная> in <список значений>
do <список команд>
done
```

Ключевые слова `for`, `do`, `done` пишутся с начала строки.

Пример, измененный сценарий `makelist`:

```
for file in adminpeople, hardpeople, softpeople
do
  sort +1 -2 $file | tr -d -9 | pr -h Distribution | lpr.
done.
```

Допустимо использование метасимволов Bash в списке значений.

Пример:

```
#для всех имен, кончающихся на people
for file in *people
do
...
done.
```

Если `in` опущено, то по умолчанию в качестве списка значений берется список аргументов процедуры, в которой содержится цикл, а если цикл не в процедуре, то список параметров командной строки (то есть в качестве процедуры выступает команда).

Пример:

```
for file
do
...
done
```

Для вызова `makelist`, `adminpeople`, `hardpeople`, `softpeople` будет сделано то же самое.

4.3.8. Оператор цикла `while`

Синтаксис оператора цикла `while`:

```
while <команда>
```

```
do
  <команды>
done
```

Если <команда> выполняется успешно, то выполнить <команды>, завершаемые ключевым словом done.

Пример:

```
if [ $# -eq 0 ]
then echo ”Usage: $0 file ...” > &2
   exit
fi
while [ $# -gt 0 ]
   do if test -s $1
   then echo ”no file $1” > &2
   else sort + 1 - 2 $1 | tr -d ... #процедуры)
   fi
   shift #перенумеровать аргументы
done
```

Процедуры выполняются над всеми аргументами.

4.3.9. Оператор цикла until

Оператор цикла until инвертирует условие повторения по сравнению с while.

Синтаксис оператора цикла until:

```
until <команда>
do
<команды>
done
```

Пока <команда> не выполнится успешно, выполнять <команды>, завершаемые словом done.

Пример:

```
if test S# -eq 0
then echo ”Usage $0 file...” > &2
     exit
fi
```

```
until  test  S# -eq 0
do
       if test -s $1
       then echo ”no file $1” > &2
       else sort +1 -2 $1 | tr -d ... #процедура
       fi
       shift                  #сдвиг аргументов
done
```

4.3.10. Оператор выбора `case`

Синтаксис оператора выбора `case`:

```
case in
string1) <если string = string1, то выполнить все следующие
команды до ;; > ;;
string2) <если string = string2, то выполнить все следующие
команды до ;; > ;;
string3) ... и т.д. ...
esac
```

Пример:

Пусть процедура имеет первый параметр `-t`:

```
.................
together = no
case  $1  in
 -t)  together = yes
   shift ;;
 -?)  echo  ”$0: no option $1”
   exit ;;
esac
if test $together = yes
  then sort ...
fi
```

где `?` — метасимвол (если `-?`, т. е. «другая» опция, отличная от `-t`, то ошибка). Можно употреблять все метасимволы языка Bash, включая `?`, `*`, `[-]`.

4.3.11. Сдвиг списка параметров

Оператор `shift` очень полезен при разборе аргументов внутри сценария и выполняет сдвиг аргументов влево с потерей первого аргумента. Первый параметр теряется, второй параметр становится первым, третий — вторым и т. д.

4.3.12. Массивы

В версии Bash, входящей в состав ОС, поддерживаются массивы. Массивы в Bash динамические, то есть в качестве ключа массива можно использовать как числа, так и строки (ассоциативные массивы).

Формат инициализации массива:

```
<имя массива>[<ключ>]=<значение>
```

Для того, чтобы воспользоваться ассоциативным массивом, нужно его сначала объявить:

```
declare -A <имя массива>
```

Примеры:

```
a[1]=1
a[2]=2
a[3]=3
declare -A dict
dict[abc]=word1
dict[cba]=word2
```

Формат использования массива:

```
${<имя массива>[<ключ>]}
```

Чтобы обратиться ко всем элементам массива сразу, нужно в качестве порядкового номера элемента массива указать символ «*» или символ «@». Например:

```
$ echo ${arr[*]}
element1 element elementN
```

Символ «@» нужно использовать, если значение хотя бы одного элемента массива может содержать пробелы. Разница между `${arr[@]}` и `${arr[*]}` такая же, как между `$@` и `$*`.

Оболочка позволяет инициализировать массив сразу с несколькими значениями. Например:

```
$ arr=(1 2 3 4 5 6 7 8 9)
```

Данный пример будет эквивалентен списку:

```
$ arr[0]=1; arr[1]=2; arr[2]=3; arr[3]=4; arr[4]=5;
arr[5]=6; arr[6]=7; arr[7]=8; arr[8]=9
```

Доступен также следующий способ инициализации массива:

```
$ mass=([0]=element0 [5]=element5)
```

Для того чтобы посмотреть количество элементов в массиве, необходимо выполнить команду:

```
$ echo ${#arr[@]}
```

Вывести содержимое всего массива:

```
$ echo ${arr[@]}
```

4.3.13. Математические выражения

В сценариях доступно два варианта для вычисления математических выражений:

– внешняя команда `expr`;

– внутренняя команда командной оболочки `let`.

Команда `expr` принимает в качестве аргументов математическое выражение и выводит результат на терминал (результат выполнения можно присвоить какой-нибудь переменной). Команда `let` принципиально отличается от `expr`: она ничего не выводит на экран, а лишь выполняет математические операции и интегрирована непосредственно в командную оболочку.

4.3.13.1. `expr`

Вычисляет выражение и выводит результат на терминал. Особенностью `expr` является обязательное разделение всех операндов и символов операций пробелами, в том числе и для скобок.

Формат команды:

```
expr <выражение>
```

Список возможных выражений `expr` приведён в таблице 4.

Таблица 4 – Список выражений `expr`

Выражение	Описание
`A | B`	A, если это не 0 иил null, в противном случае B

Выражение	Описание
`A & B`	A, если B не 0 или null, в противном случае 0
`A < B`	1, если A меньше, чем B, 0 иначе
`A <= B`	1, если A меньше или равно B, 0 иначе
`A = B`	1, если A равно B, 0 иначе
`A != B`	1, если A не равно B, 0 иначе
`A >= B`	1, если A больше или равно B, 0 иначе
`A > B`	1, если A больше, чем B, 0 иначе
`A + B`	Арифметическая сумма A и B
`A - B`	Арифметическая разность A и B
`A * B`	Арифметическое произведение A и B
`A / B`	Арифметическое частное от деления A на B
`A % B`	Арифметический остаток от деления A на B
`СТРОКА : РЕГ`	1, если СТРОКА соответствует регулярному выражению РЕГ
`match СТРОКА РЕГ`	Аналогично `СТРОКА : РЕГ`
`substr СТРОКА ПОЗИЦИЯ ДЛИНА`	Выводит подстроку строки СТРОКА длины ДЛИНА, начиная с позиции ПОЗИЦИЯ
`index СТРОКА СИМВОЛЫ`	Если в строке СТРОКА есть хотя бы один символ из набора СИМВОЛЫ, выводит позицию первого совпадающего символа
`length СТРОКА`	Возвращает длину строки
`+ ВЫРАЖЕНИЕ`	Интерпретировать ВЫРАЖЕНИЕ как обычную строку, даже если это ключевое слово
`( ВЫРАЖЕНИЕ )`	Значение выражения в скобках

Примеры:

```
$ expr 2 + 3
5
$ expr (2 + 3) + 2 //неправильно!
Bash: syntax error near unexpected token ‘2’
$ expr ( 2 + 3 ) + 2 //неправильно!
Bash: syntax error near unexpected token ‘2’
```

```
    $ expr \( 2 + 3 \) + 2 //правильный вариант: с
экранированием специальных символов командной оболочки и
пробелами
    7
    $ expr 1 = 2
    0
    $ expr 1 = 1
    1
    12
    $ expr substr 123 1 2
    12
    $ expr substr 123 2 2
    23
    $ expr 1 | 2 //нерпавильно, нужно экранирование
    Bash: 2: команда не найдена
    $ expr 1 \| 2 //правильно
    1
    $ expr 0 \| 2
    2
    $ expr 0 \& 2
    0
    $ expr 2 \& 3
    2
    $ expr 2 \& 0
    0
    $ expr 1 \& 0
    0
    $ expr 5 / 0
    expr: деление на ноль
    $ expr 5 / 3
    1
    $ expr index 123 12
    1
    $ expr index 123 32
```

```
2
$ expr index 123 3
3
$ expr index 123 5
0
```

Запись результата вычисления выражения в переменную осуществляется командой вида:

```
$ a=‘expr 2 + 2‘
$ echo $a
4
```

Пример суммирования трех чисел:

```
$ cat sum3
expr $1 + $2 + $3
$ chmod 755 sum3
$ ./sum3  13  49  2
64
```

Пример непосредственного использования команды в терминале:

```
$ expr 13 + 49 + 2 + 64 + 1
129
```

Знак умножения следует экранировать (`\*`) или заключать в кавычки (одинарные или двойные), например: `'*'`, так как символ «*» имеет в Bash специальный смысл.

Применение команды `expr` в процедуре (фрагмент):

```
num=‘wc -l< $1‘
tot=100
count=$num
avint=‘expr $tot / $num‘
avdec=‘expr $tot % $num‘
while test $count -gt 0
do ...
```

Здесь `wc -l` осуществляет подсчет числа строк в файле, а далее это число используется в выражениях.

4.3.13.2. `let`

Команда `let` не является отдельной программой, а интерпретируется непосредственно командной оболочкой. Её синтаксис несколько отличается от `expr`, так как `let` ничего не выводит на терминал, однако `let` позволяет выполнять точно такие же математические выражения.

Формат команды:

```
let <выражение>
```

или

```
((выражение))
```

Так как `let` не выводит результат на терминал, для того чтобы увидеть результат вычисления, необходимо присвоить его какой-либо переменной при помощи одной из операций присваивания, после чего вывести содержимое полученной переменной на экран.

Список математических операций `let` приведён в таблице 5.

Таблица 5 – Список выражений `let`

Выражение	Описание
`A++` или `A--`	Постфиксный инкремент или декремент переменной
`++A` или `--A`	Префиксный инкремент или декремент переменной
`+A` или `-A`	Унарный плюс или минус
`!A`	Логическое отрицание
`~A`	Побитовое отрицание
`A**B`	Возведение `A` в степень `B`
`A*B`	Произведение `A` и `B`
`A/B`	Частное от деления `A` на `B`
`A%B`	Остаток от деления `A` на `B`
`A+B`	Сумма `A` и `B`
`A-B`	Разность `A` и `B`
`A>>B` или `A<<B`	Правый или левый побитовый сдвиг
`A==B`	1, если `A` равно `B`
`A!=B`	1, если `A` не равно `B`
`A>B`	1, если `A` больше `B`
`A>=B`	1, если `A` больше или равно `B`

Выражение	Описание
A<B	1, если A меньше B
A<=B	1, если A меньше или равно B
A&B	Побитовое «И»
A^B	Побитовое исключительное «ИЛИ»
A|B	Побитовое «ИЛИ»
A&&B	Логическое «И»
A||B	Логическое «ИЛИ»
A?B:C	Если A — истина, то B, иначе C
A=B	Присвоить A значение переменной B
A*=B	Присвоить A значение выражения A*B
A/=B	Присвоить A значение выражения A/B
A%=B	Присвоить A значение выражения A%B
A+=B	Присвоить A значение выражения A+B
A-=B	Присвоить A значение выражения A-B
A<<=B	Присвоить A значение выражения A<<B
A>>=B	Присвоить A значение выражения A>>B
A&=B	Присвоить A значение выражения A&B
A|=B	Присвоить A значение выражения A|B
A^=B	Присвоить A значение выражения A^B
A,B	Независимо вычисление выражений A и B

Примеры:

```
$ let 2+2 //let не выводит ничего на экран
$ echo $? //код возврата сообщает только об успешности вычисления выражения
0
$ let a=2+2 && echo $a //вывод результата на экран
4
$ ((a=2*2,b=2+2)) && echo $a $b
4 4
```

4.3.14. Интерактивный ввод данных пользователя

Команда `read` представляет собой удобный инструмент диалога с пользователем. Она считывает ввод со стандартного потока ввода и передаёт его сценарию. Это может быть как строка, введённая пользователем с клавиатуры, так и вывод другой программы или содержимое файла при использовании сценария в конвейере.

`read` последовательно присваивает поток ввода переменным, указанным в качестве аргументов. Если число слов в строке больше, чем аргументов команды `read`, в последней переменной окажется вся оставшаяся часть строки.

Пример.

Содержимое сценария `read.sh`:

```
#!/bin/sh
while read a b c
do
    echo a=$a
    echo b=$b
    echo c=$c
done
```

Пример работы со сценарием:

```
$ sh read.sh
1 2 3
a=1
b=2
c=3
1 2 3 4
a=1
b=2
c=3 4
1234
a=1234
b=
c=
<Ctrl+D>
```

4.3.15. Обработка прерываний в процедурах

Если при выполнении процедуры получен сигнал прерывания (например, от клавиши [Break] или [Del]), то все созданные временные файлы останутся не удаленными ввиду немедленного прекращения процесса.

Обработка прерываний внутри процедуры производится оператором `trap`. Синтаксис оператора `trap`:

```
trap 'command arguments' signals...
```

Кавычки формируют первый аргумент из нескольких команд, разделенных точкой с запятой. Они будут выполнены, если возникнет прерывание, указанное аргументами `signals` (целые):

– `2` — прерывание процесса;

– `1` — «зависание» процесса.

Пример:

```
case $1 in
.....
*) trap 'rm /tmp/*; exit' 2 1 (удаление временных файлов)
if test -s $1
   ..............
   rm /tmp/*
```

4.3.16. `xargs` — формирование аргументов

`xargs` — утилита для формирования списка аргументов и выполнение команды. Команда `xargs` объединяет зафиксированный набор заданных в командной строке начальных аргументов с аргументами, прочитанными со стандартного ввода, и выполняет указанную команду один или несколько раз.

Формат команды:

```
xargs [<параметры>] [<команда> [<аргумент...>]
```

Параметры:

– `-I<строка>` — выполняет поиск и замену строки на аргумент, при этом команда будет выполенена для каждой строки стандартного ввода;

– `-l[<число>]` — выполнять команду для заданного числа прочитанных со стандартного ввода аргументов, по умолчанию 1;

– `-n` — разбивать аргументы по заданным порциям;

– `-t` — печатать команду на экране перед тем как её выполнить;

– `-p` — интерактивный режим: перед выполнением каждой команды пользователь должен будет дать утвердительный ответ;

– `-0` — разбивать поток ввода не по символам перевода строки, а по нулевому символу.

4.3.16.1. Примеры

Перемещает все файлы из каталога `$1` в каталог `$2` и сообщает о каждом перемещении перед тем, как её выполнить:

```
ls $1 | xargs -I {} -t mv $1/{} $2/{}
```

Объединяет вывод команд, заключенных в скобки, в одну строку, которая затем добавляется в конец файла `log`:

```
(logname; date; echo $0 $*) | xargs >> log
```

Применяет команду `diff` к последовательным парам своих аргументов.

```
echo $* | xargs -n2 diff
```

Удаляет все файлы из временного каталога, но при этом если в именах будут пробелы, команда завершится неудачей:

```
find /tmp -name core -type f -print | xargs /bin/rm -f
```

Аналог предыдущей команды, но разбиение файлов осуществляется по нулевому символу и даже файлы с пробелами будут обработаны без ошибок, данный приём полезно использовать всегда для корректной сложной обработки файлов:

```
find /tmp -name core -type f -print0 | xargs -0 /bin/rm -f
```

4.3.17. Отладка процедур Bash

В ОС имеются три средства, позволяющие вести отладку процедур Bash:

– размещение в теле процедуры команд `echo` для выдачи сообщений, являющихся трассой выполнения процедуры;

– параметр `-v` в команде Bash приводит к печати команды на экране перед ее выполнением;

– параметр `-x` в команде Bash приводит к печати команды с раскрытием всех выражений и подстановкой значений переменных — режим отладки. Все отладочные сообщения выводятся с символом «+», что позволяет отличать их от обычных сообщений.

Пример.

Сценарий, находящийся в файле `test.sh`:

```
#!/bin/sh
[ ”$1” = ”a” ] && echo ”Задан параметр a”
[ ! ”$1” = ”b” ] && echo ”Не задан параметр b”
```

Различные варианты исполнения:

```
$ sh test.sh a
Задан параметр a
Не задан параметр b
$ sh test.sh b
$ sh -v test.sh
#!/bin/sh
[ ”$1” = ”a” ] && echo ”Задан параметр a”
[ ! ”$1” = ”b” ] && echo ”Не задан параметр b”
Не задан параметр b
$ sh -v test.sh a
#!/bin/sh
[ ”$1” = ”a” ] && echo ”Задан параметр a”
Задан параметр a
[ ! ”$1” = ”b” ] && echo ”Не задан параметр b”
Не задан параметр b
$ sh -v test.sh b
#!/bin/sh
[ ”$1” = ”a” ] && echo ”Задан параметр a”
[ ! ”$1” = ”b” ] && echo ”Не задан параметр b”
$ sh -x test.sh
+ ’[’ ’’ = a ’]’
+ ’[’ ’!’ ’’ = b ’]’
+ echo ’Не задан параметр b’
Не задан параметр b
$ sh -x test.sh a
+ ’[’ a = a ’]’
+ echo ’Задан параметр a’
Задан параметр a
+ ’[’ ’!’ a = b ’]’
```

```
+ echo 'Не задан параметр b'
Не задан параметр b
$ sh -x test.sh b
+ '[' b = a ']'
+ '[' '!' b = b ']'
```

4.4. Основные команды Bash

В данном разделе перечислено краткое описание основных команд. Подробное описание, синтаксис и примеры использования данных команд можно найти в соответствующих разделах данного руководства. Перечень основных команд приведён в таблице 6.

Таблица 6 – Перечень основных команд Bash

Команда	Краткое описание	Раздел
alias	Создание псевдонимов команд	4.2.8.4
anacron	Планировщик задач с периодическим выполнением	8.4.2
at, batch	Выполнение задач в заданное время	**??**
awk	Выполняет операции над строками по шаблону	7.5.29
badblocks	Анализ диска	7.4.8
basename	Выделение имени файла из полного пути	7.3.20
bash, sh	Командная оболочка Bash	4.2
blkid	Получение уникального идентификатора дискового раздела	7.4.5
cat	Выводит содержимое файла в терминал	7.5.1
cal	Календарь	4.7.2
cd	Изменяет текущий каталог. Без параметра — переход в домашний каталог	7.3.8
chattr	Изменение расширенных атрибутов файла	**??**
chgrp	Изменение группы владельца файла	**??**
clear	Очистка экрана	4.2.8.8
chfn	Изменение фамилии, имени, отчества пользователя и его контактной информации	6
chmod	Изменение дискреционных прав доступа на файл	**??**
chown	Изменение владельца файла	**??**

Команда	Краткое описание	Раздел
cmp	Сравнивает два файла	7.5.10
cp	Копирует файлы и каталоги	7.3.9
crontab	Планирование задач	8.4.1
cut	Выделение столбцов из строк	7.5.12
date	Настройка даты и времени	4.7.1
dd	Копирование файлов и блочных устройств	7.9.1
df	Проверка свободного места на дисках	7.4.12
diff	Сравнивает два файла и выводит список изменений, которые нужно внести в первый файл, чтобы получить в точности второй файл	7.5.9
du	Вычисление объёма дискового пространства, занимаемого файлами	7.4.14
echo	Выводит сообщение на терминал	7.5.7
ed	Интерактивный текстовый редактор	7.5.27
egrep	Фильтрация с использованием расширенных регулярных выражений, аналог `grep -E`	7.5.25
eject	Извлечение лотка привода компакт-дисков	7.4.15
env	Получение и установка переменных окружения	4.2.7.3
exit	Выход из командной оболочки или сценария	4.2.8.10
export	Установка переменных окружения	4.2.7.2
expr	Вычисление выражений	4.3.13.1
fdisk	Утилита для разбиения диска на разделы	7.4.7
fgrep	Фильтрация строк из одного файла строками из другого файла, аналог `grep -F`	7.5.26
file	Просмотр информации о типе файла	7.3.19
find	Поиск файлов по заданному критерию	7.3.14
findmnt	Поиск примонтированных файловых систем	7.4.13
free	Информация о использовании памяти	9.1.2.5
fsck	Проверка состояния файловых систем	7.4.11
fuser	Идентификация использования ресурсов процессами	8.1.10
getfacl	Получить информацию о списках доступа	**??**

Команда	Краткое описание	Раздел
grep	Фильтрация строк по заданному регулярному выражению, поиск по содержимому файлов	7.5.24
head	Кусочный вывод файла или вывода команды с начала	7.5.5
history	Просмотр и очистка истории команд	4.2.8.2
iconv	Конвертация кодировок текстовых файлов	7.5.22
ifconfig	Диагностика и настройка сети	ч. 2
ionice	Изменяет приоритет ввода-вывода	8.1.6
iptables	Межсетевой экран	ч. 2
killall	Отправка процессу сигнала по имени	8.1.2
kill	Отправка процессу сигнала	8.1.1
less	Чтение файлов или вывода команд с прокруткой	7.5.2
let	Математические операции над переменными Bash	4.3.13.2
ln	Создание жёстких и символьных ссылок	7.3.6
lsof	Информация об использовании процессами файлов	8.1.11
ls	Просмотр содержимого каталога, по умолчанию выводит содержимое текущего каталога	7.3.1
lsattr	Просмотр расширенных атрибутов	7.3.2
lsblk	Список блочных устройства	7.4.6
lscpu	Информация о процессорах	9.1.2.12
lspci	Информация об устройствах PCI	9.1.2.11
lspcmcia	Информация об устройствах PCMCIA	**??**
lsusb	Информация об устройствах USB	9.1.2.10
lvm	Менеджер логических томов	7.8
mail	Отправка и чтение почты	6.1.3
man	Встроенная справочная система	4.2.8.7
mesg	Настройка приёма сообщений от других пользователей	6.1.2
mkdir	Создаёт каталог	7.3.3
mkfifo	Создаёт именованный канал	7.3.13
mkfs	Создание файловых систем	7.4.4
mknod	Создаёт файл устройства	7.3.4

Команда	Краткое описание	Раздел
more	Листание файла от начала до конца, предназначена для чтения длинных файлов или вывода команд, который не помещается на экране	7.5.3
mount	Монтирует файловые системы	7.4.2
mv	Перемещает или переименовывает файлы и каталоги	7.3.10
nice	Запуск процесса с заданным приоритетом	8.1.5
passwd	Изменение пароля пользователя	6
paste	Слияние строк файлов	7.5.13
pg	Просмотр файлов на экране	**??**
pgrep	Поиск процессов	8.1.8
ping	Проверка доступности сети	ч. 2
pkill	Утилита для уничтожения процессов	8.1.9
poweroff	Выключить питание компьютера	ч. 2
pr	Форматная печать файлов	7.5.4
printenv	Вывод переменных окружения	4.2.7.3
printf	Вывод форматированных строк	7.5.8
pstree	Просмотр списка процессов в виде дерева	9.1.2.8
ps	Просмотр списка процессов	9.1.2.4
pwd	Выводит текущий рабочий каталог	7.3.7
reboot	Перезагрузить компьютер	ч. 2
renice	Изменение приоритета уже запущенного процесса	8.1.7
rmdir	Удаляет каталоги	7.3.12
rm	Удаляет файлы	7.3.11
route	Настройка маршрутизации	ч. 2
rpm	Установка и удаление пакетов, просмотр информации о пакетах	5.3
run_init	Управление службами с сохранением контекста безопасности процессов. Для управления службами должна использоваться эта команда	8.3.1
sed	Потоковый текстовый редактор	7.5.28
seq	Генерирует последовательность чисел	4.6.1

Команда	Краткое описание	Раздел
`service`	Запуск и остановка системных служб. Не рекомендуется к постоянному использованию, так как не восстанавливает контекст безопасности (см. `run_init`)	8.3.1
`setfacl`	Настроить списки доступа	**??**
`showmount`	Просмотр информации о состоянии NFS	ч. 2
`sleep`	Пауза в секундах	8.1.3
`sort`	Сортировка строк	7.5.16
`split`	Разбить файл на части по строкам или по размеру	7.5.18
`stat`	Просмотр информации о файле	7.3.18
`tail`	Кусочный вывод файла или вывода команды с конца	7.5.6
`tar`	Архиватор	7.7.1
`tee`	Сохранение потока стандартного ввода в файл	7.5.19
`test`	Проверка логических выражений	4.3.6
`time`	Вычисление времени выполнения команды	4.7.3
`top`	Интерактивный монитор процессов	9.1.2.1
`touch`	Создаёт файл или обновляет временную метку уже существующего	7.3.5
`tr`	Замена символов	7.5.20
`traceroute`	Трассировка маршрута до узла в сети	ч. 2
`type`	Выполняет поиск исполняемых файлов	7.3.16
`umount`	Размонтирование файловых систем	7.4.3
`unalias`	Удаление псевдонимов команд	4.2.8.4
`uniq`	Удаляет повторяющиеся строки	7.5.15
`uptime`	Информация о времени работы компьютера	9.1.2.9
`useradd`	Создание пользователя	6
`userdel`	Удаление пользователя	6
`usermod`	Изменение параметров существующего пользователя.	6
`usleep`	Пауза в микросекундах	8.1.4
`uuidgen`	Создаёт уникальный универсальный идентификатор	4.6.3
`vi`, `vim`	Универсальный текстовый редактор	4.8
`wc`	Подсчёт количества строк, символов, слов	7.5.14
`who`	Список пользователей, работающих в системе	9.1.2.7

Команда	Краткое описание	Раздел
whereis	Информация о расположении файлов	7.3.17
which	Выполняет поиск исполняемых файлов	7.3.15
w	Список пользователей, работающих в системе и их действия	9.1.2.6
write	Отправка сообщений другим пользователям	6.1.1
xargs	Формирует из стандартного ввода строку аргументов для команды	4.3.16
yes	Непрерывно выводит строку	4.6.2
yum	Установка и удаление пакетов с учётом зависимостей	5.1

П р и м е ч а н и е. Пометкой «ч. 2» обозначаются команды, описанные во второй части руководства администратора.

4.5. Менеджер окон в терминале screen

Утилита screen — полноэкранный функциональный консольный оконный менеджер с поддержкой прокрутки, поиска в окне, функцией копирования-вставки между окнами. Важной особенностью является возможность отсоединиться от screen, завершить сеанс работы в Bash, после чего заново восстановить предыдущую сеанс screen.

Утилита screen является важным инструментом системного администратора, так как выполняет функции оконного менеджера при отсутствии графического интерфейса, что важно при управлении системами, например, по SSH. Основные возможности утилиты:

1) создание и управление несколькими виртуальными окнами в одном единственном окне терминала: переключение между окнами, отслеживание активности в каждом окне, обмен содержимым между окнами с помощью копирования и вставки, назначение заголовков окнам;

2) разделение рабочего пространства терминала на несколько частей и переключение между ними. Так администратор может разделить рабочее пространство, чтобы наблюдать за несколькими терминалами одновременно;

3) сохранение сеанса работы даже при обрыве соединения по SSH или отключении от терминала с возможностью возобновить работу в терминале;

4) возможность настройки горячих клавиш.

4.5.1. Конфигурационный файл

Для настройки программы используется конфигурационный файл `.screenrc`, который находится в домашнем каталоге. Если его там нет, можно скопировать файл системный файл `screenrc` который находится в каталоге `/etc`.

Все параметры можно изменить во время работы. Для этого нажмите сочетание клавиш [Ctrl+a :] и введите название параметра и его значение.

Некоторые директивы.

`vbell off` управляет визуальным звонком. Если данный параметр будет включен (`on`) то звонок будет отображаться, как вспышка на экране.

`activity 'activity in window %n'` сообщение, которое будет выводиться при включенном режиме мониторинга за окном.

`bell_msg 'bell in window %n'` сообщение, которое выведется на экран в случае получения звукового сигнала в каком-либо окне.

`nethack on` изменяет стиль текста выводимых сообщений.

`autodetach on` если по какой-то причине соединение с управляющим процессом будет потеряно, то после восстановления работа в `screen` может быть возобновлена. В обратном случае (`off`) — `screen` будет уничтожен со всеми дочерними окнами и процессами.

`startup_message off` выключает сообщение об авторских правах при первом запуске.

`defscrollback 10000` количество строк по умолчанию для буфера прокрутки.

`caption always` показывает заголовки окна в строке статуса.

`caption string "%{rk} %c %{dd} %{+b M}%n %{-b dd}%-w%{+b B.}%n* %t%{-}%+w%<"` форматирование строки статуса. Данный набор символов приведет к тому, что в строке статуса будет отображаться время и цветом выделяться активное окно.

4.5.2. Комбинации клавиш при работе

Все комбинации в `screen` начинаются с нажатия управляющей последовательности [Ctrl+a], поэтому во всех сочетаниях ниже она опущена для наглядности. Для того чтобы отправить сочетание клавиш [Ctrl+a] какой-либо программе внутри

screen (например, это может быть screen внутри screen), используйте сочетание [Ctrl+a a].

Выход из screen:

– d — отсоединение от сессии, при этом сессия и все работающие внутри неё приложения сохраняются и не завершаются;

– DD — отсоединиться и выйти (быстрый выход), например, если был произведён вход по SSH, данное сочетание закроет сеанс SSH, но сохранит сеанс screen;

– :quit или :exit — выйти из сеанса screen (закрыть сеанс).

Получение справки:

– ? — вывести список сочетаний клавиш.

Управление окнами:

– c — создание нового окна;

– a — переключение между окнами — используется для быстрого перемещения между двумя последними окнами;

– <НОМЕР> — выбор окна по номеру (от 0 до 9);

– p — предыдущее окно;

– n — следующее окно;

– " — список окон для переключения;

– '<заголовок или номер> — перейти к окну по заголовку или номеру;

– A — изменить заголовок окна;

– C — очистить окно;

– F — подогнать размер окна под текущий размер терминала;

– K — уничтожить окно (опасное действие, не рекомендуется использовать для повседневной работы);

– \ — уничтожить все окна (опасное действие, не рекомендуется использовать для повседневной работы);

– r — переключение режима переноса по словам;

– b — отображение заголовка окна (по умолчанию заголовки скрыты);

Разделение окон:

– S — текущее окно разделяется на две области по горизонтали и в обоих можно открыть по новому окну;

– | или V — разделить текущее окно на две области по вертикали;

– X — закрыть текущую область;

– Q — закрыть все области кроме текущей;

– TAB — переключение между разделенными частями окна.

Разное:

– [или ESC — режим прокрутки, он же режим копирования. Для копирования подведите курсор к нужному месту и нажмите пробел;

–] — вставка выделенной области;

– H — протоколирование окна в файл screenlog;

– M — режим слежения за активностью в окне;

– _ — режим слежения за неактивностью в окне;

– x — «заблокировать» менеджер, для того чтобы разблокировать менеджер, нужно будет ввести пароль пользователя.

4.5.3. Параметры

-rd подключиться к screen. Сделать deatach для остальных сессий;

-ls список запущенных менеджеров;

-dm запуск screen в режиме deatach;

-wipe удалить сведения о запущенных менеджерах;

-x присоединиться к screen.

-S <сессия> создать новую сессию с именем <сессия>.

-r [сессия] присоединиться к указанной сессии или последней сессии. Список сессий можно узнать с помощью параметра -ls.

Можно использовать возможность автоматического дополнения совместно с параметром -r. Чтобы не вводить длинное имя сессии, введите:

```
screen -r [TAB][TAB]
```

Система автоматически дополнит имя сессии или, если сессий несколько, выведет варианты дополнения на экран.

4.5.4. Примеры использования

Утилиту screen удобно использовать для запуска какой-либо долго выполняющейся задачи в фоновом режиме. Данная проблема актуальна в связи с тем, что Bash завершает все дочерние процессы и все задачи, связанные с текущим терминалом, когда пользователь покидает данный терминал. Иногда пользователь может случайно закрыть терминал или, в случае соединения по SSH, может оборваться текущее соединение, что в итоге приведёт к тому, что важная длительная операция, например, резервное копирование, завершится без предупреждения.

Утилита screen не завершает задачу, даже если пользователь закрыл терминал или закрыл соединение SSH, поэтому в случае работы по SSH, где существует риск обрыва соединения, рекомендуется всегда использовать screen.

Пример использования screen для выполнения длительной операции:

```
# screen
# wget http://server.local/cd-image.iso
```

Нажмите сочетание клавиш [Ctrl+a d], чтобы выйти:

```
[detached]
#
```

После этого можно закрыть сессию SSH или Bash, выполнив команду exit или запустить ещё один сеанс screen.

Проверить статус задач:

```
# screen -ls
There is a screen on:
        20673.pts-0.srv (Detached)
1 Socket in /var/run/screen/S-root.
```

Можно снова присоединиться к данному терминалу:

```
# screen -r 20673.pts-0.srv
```

Или, если это был единственный терминал:

```
# screen -r
```

Таким образом, screen позволяет управлять сразу несколькими длительными по времени задачами без риска случайного завершения.

4.6. Утилиты-генераторы

Большинство команд Bash являются обработчиками — они принимают какой-либо ввод от пользователя и выполняют его обработку: поиск, сортировка, выборка и т. п. В данном разделе перечислены утилиты, которые являются генераторами данных, не требуя никаких входных файлов. Утилиты-генераторы полезны при создании тестовых файлов и при использовании внутри сценариев Bash.

Наиболее часто используемые утилиты — echo и printf. Утилита echo просто выводит аргументы в окно терминала, автоматически добавляя перевод строки. Команда printf работает аналогично функции printf языка C. Некоторые примеры использования:

```
$ echo ”Hello world”
```

```
Hello world
$ echo -e ”\tHello world”
    Hello world
$ echo -en ”\tHello world\n”
    Hello world
$ echo test$_{1..10}
test test test test test test test test test test
$ printf ”Hello world\n”
Hello world
$ printf ”Hello, %s\n” username
Hello, username
$ printf ”%s, %s\n” Bye username
Bye, username
```

Команда `printf` полезна для формирования вывода по столбцам:

```
$ printf ”%10s %-10s %s\n” a b c
         a b          c
$ printf ”%10s %-10s %s\n” abba bacc cabb
      abba bacc       cabb
```

Сама командная оболочка Bash содержит несколько генераторов, которые иногда бывают очень полезны.

Пример генерации чисел:

```
$ echo {1..10}
1 2 3 4 5 6 7 8 9 10
```

Пример генерации чисел от 1 до 1000 и их суммирование:

```
$ echo {1..1000} | tr ’ ’ ’+’ | bc
500500
```

Генерация строк из нескольких частей:

```
$ echo {a,b,c}{d,e,f}
ad ae af bd be bf cd ce cf
```

Можно генерировать повторяющиеся несколько раз строки, например:

```
$ echo test$_{1..5}
test test test test test
```

Перемножить 10 раз число 1024:

```
$ echo 1024$_{1..10} | tr ’ ’ ’*’ | bc
1267650600228229401496703205376
```

Генерация надёжных паролей пользователей нередко входит в обязанности системного администратора. Данный процесс можно автоматизировать с помощью устройства, представляющего собой датчик псевдослучайных чисел `/dev/urandom`. Для массовой генерации паролей можно использовать одну из следующих команд:

```
# base64 /dev/urandom -w10 | head -20
# < /dev/urandom tr -dc A-Z-a-z-0-9 | fold -w10 | head -20
```

Здесь параметр `-w10` означает, что длина пароля равняется десяти символам, а `head -20` — количество паролей (20 штук).

Однако следует учитывать, что полученные таким образом пароли могут быть слишком сложны для запоминания, а это может привести к тому, что пользователи будут их записывать и хранить в каком-нибудь легкодоступном месте, например на своём рабочем столе. Системный администратор должен предвидеть такой вариант развития событий и проводить соответствующий инструктаж персонала.

4.6.1. `seq`

Утилита `seq` генерирует последовательность чисел.

Формат команды:

`seq [<параметры>] M`

Генерирует последовательность от 1 до `M`.

`seq [<параметры>] N M`

Генерирует последовательность от `N` до `M`.

`seq [<параметры>] N S M`

Генерирует последовательность от `N` до `M` с шагом `S`.

Параметры:

`-f, –format <формат>` использовать `<формат>` как у функции `printf` в языке C.

`-s, –separator <строка>` использовать строку для разделения выводимых чисел (по умолчанию перевод строки).

`-w, –equal-width` выровнять ширину, заполнив пробелы нулями слева.

Если параметры `N` или `S` пропущены, то считается, что они равны 1, даже если `M` меньше 1. Параметры `N`, `S` и `M` воспринимаются как числа с плавающей запятой. Параметр `S` должен быть положительным, если `N` меньше, чем `M`, и, соответственно

отрицательным, если больше. Параметр <формат> должен содержать только один из шаблонов вывода printf: %e, %f, %g.

4.6.2. yes

Непрерывно (пока пользователь принудительно не завершит программу) выводит строку.

Формат команды:

```
yes [<строка>]
```

Без параметров выводит символ «y».

Хотя с первого взгляда она может показаться бесполезной, команда yes может пригодиться для превращения интерактивных команд в групповые. Если нужно избавиться от надоедливого сообщения «Вы уверены, что хотите сделать это?», передайте выходные данные команды yes на вход такой программе, чтобы автоматически утвердительно ответить на все вопросы.

Пример:

```
$ yes | very_talkie_program
```

При помощи данной программы можно создать очень быстрорастущий файл:

```
$ yes > BIG_file
```

Загрузить центральный процессор:

```
$ yes > /dev/null
```

4.6.3. uuidgen

Данная команда генерирует UUID (универсальный уникальный идентификатор) случайным образом. UUID создаётся таким образом, чтобы быть уникальным не только на данном компьютере, но и на других машинах, а также быть уникальным среди тех UUID, которые будут созданы в будущем. UUID может применяться для различных целей: идентификация методов в библиотеке или идентификация устройств и носителей.

Формат команды:

```
uuidgen [-r|-t]
```

Если задан параметр -r, создаётся полностью случайный UUID, основанный на датчике случайных чисел, если задан параметр -t — создаётся UUID, основанный на текущем времени. Если параметр не задан, команда автоматически выбирает наилучший метод.

4.7. Время UNIX

Время UNIX стандарт описания времени, принятый в UNIX и других POSIX-совместимых операционных системах, включая ОС. Определяется как количество секунд, прошедших с полуночи 00:00:00 1 января 1970 года; время с этого момента называют «эпохой UNIX».

Представление времени в виде количества секунд удобно использовать для сравнения и хранения дат (дата и время в этом формате занимают всего 4 или 8 байтов). При необходимости обращения к элементам дат (день, месяц, год) секунды можно преобразовать в любой подходящий формат (и наоборот).

Данный формат представляется удобным для системных администраторов, так как позволяет легко хранить, считывать и сравнивать значения дат. Многие утилиты ОС используют данный формат времени, что будет указано явно.

В программах для хранения времени UNIX используется целочисленный знаковый тип. Знаковость упрощает вычисление разницы в секундах между двумя моментами времени, которая может быть отрицательной. 32-битные числа со знаком могут ссылаться на моменты времени от пятницы 13 декабря 1901 года 20:45:52 до вторника 19 января 2038 года 03:14:07 включительно.

4.7.1. `date`

Утилита `date` позволяет устанавливать время, форматировать его и отображать текущую дату.

Формат команды:

```
date [<параметры>] [+<формат>]
```

или

```
date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
```

Выводит текущее время в заданном формате или устанавливает системное время.

`-d, --date=<строка>` показать не текущее время, а время, описанное в `<строка>`.

`-f, --file=<файл>` соответствует применению `--date` для каждой строки из `<файл>`.

`-r, --reference=<файл>` показать время последнего изменения `<файл>`.

`-R, --rfc-2822` выводить дату и время в формате RFC 2822.

`--rfc-3339=date|seconds|ns` выводить дату и время в формате RFC 3339 с указанной точностью.

`-s, --set=<строка>` установить время, описанное в `<строка>`.

`-u, --utc, --universal` показать или установить универсальное координированное время.

Последовательность `<формат>` управляет выводом. Единственный ключ, допустимый для второй формы, задает координированное универсальное время. Воспринимаются следующие последовательности:

`%%` знак «%».

`%a` местное сокращенное название дня недели (например, Вск).

`%A` местное полное название дня недели (например, Понедельник).

`%b` местное сокращенное название месяца (например, Янв).

`%B` местное полное название месяца (например, Январь).

`%c` местная дата и время (например, Чтв 05 Июл 2007 11:57:29).

`%C` век; аналогично `%Y`, кроме пропуска последних двух символов (например, 21).

`%d` день месяца (например, 01).

`%D` дата; аналогично `%m/%d/%y`.

`%e` день месяца, дополненный пробелами; аналогично `%_d`.

`%F` долная дата; эквивалентно `%Y-%m-%d`.

`%g` последние две цифры года, соответствующего номеру недели в году согласно ISO 8601 (см. `%G`).

`%G` год, соответствующий номеру недели в году согласно ISO 8601 (см. `%V`); обычно имеет смысл только в сочетании с `%V`.

`%h` то же, что и `%b`.

`%H` час (00..23).

`%I` час (01..12).

`%j` номер дня в году (001..366).

`%k` час (0..23).

`%l` час (1..12).

`%m` месяц (01..12).

`%M` минута (00..59).

`%n` новая строка.

`%N` наносекунды (000000000..999999999).

`%p` местный эквивалент AM или PM; пусто если неизвестно.

`%P` аналогично `%p`, но в нижнем регистре.

`%r` местное 12-часовое время (например, 11:11:04 PM).

`%R` 24-часовой формат часов и минут; аналогично `%H:%M`.

`%s` число секунд, истекших с 1970-01-01 00:00:00 UTC (также известно как «время UNIX»).

`%S` секунда (00..60).

`%t` символ табуляции.

`%T` время; аналогично `%H:%M:%S`.

`%u` день недели (1..7); 1 — понедельник и т. п..

`%U` номер недели в году, начинающейся с воскресенья (00..53).

`%V` номер недели в году, начинающейся с понедельника, согласно ISO 8601 (01..53).

`%w` день недели (0..6), 0 означает воскресенье.

`%W` номер недели в году, начинающейся с понедельника (00..53).

`%x` местное представление даты (например, 31.12.1999).

`%X` местное представление времени (например, 23:13:48).

`%y` последние две цифры года (00..99).

`%Y` год (например, 1999).

`%z` часовой пояс в формате «+ччмм» (например, -0400).

`%:z` часовой пояс в формате «+чч:мм» (например, -04:00).

`%::z` часовой пояс в формате «+чч:мм:сс» (например, -04:00:00).

`%:::z` часовой пояс с минимально необходимым количеством двоеточий (например, -04, +05:30).

`%Z` алфавитное сокращение часового пояса (например, EDT).

По умолчанию `date` дополняет числовые поля нулями. После «`%`» могут идти следующие необязательные флаги:

– «`-`» (дефис) — не дополнять это поле;

– «_» (подчёркивание) — дополнять пробелами;

– «`0`» (ноль) — дополнять нулями;

– «`^`» — использовать верхний регистр, если возможно;

– «`#`» — использовать противоположный регистр, если возможно.

После любого из флагов идет необязательная ширина поля в виде десятичного числа; затем необязательный модификатор, один из следующих:

`E` если возможно, использовать альтернативное представление для текущей локали.

`O` если возможно, использовать альтернативные числовые символы локали.

Примеры.

Узнать, какой вчера был день:

```
$ date --date=”yesterday”
```

или

```
$ date --date=”1 days ago”
```

Узнать, какого числа пятница:

```
$ date --date=”this friday”
```

4.7.2. `cal`

Простой календарь.

Формат команды:

```
cal [<параметры>] [<день> [<месяц> [<год>]]]
```

Параметры:

`-1` отображать только один месяц (по умолчанию).

`-3` выводить календарь на три месяца: текущий, предыдущий и следующий.

`-s` отображать воскресенье в качестве первого дня недели.

`-m` отображать понедельник в качестве первого дня недели.

`-j` отображать дни в формате количества дней с 1 января.

`-y` календарь на текущий год.

4.7.3. `time`

Утилита `time` позволяет подсчитать, сколько времени проработала та или иная программа.

Формат команды:

```
time <команда> [<аргументы...>]
```

Пример:

```
$ time sleep 1
real 0m1.001s
user 0m0.000s
sys 0m0.001s
```

4.8. Универсальный текстовый редактор vi

Текстовый редактор vi — универсальный редактор операционных систем семейства UNIX. В ОС установлена как классическая версия редактора, так и редактор vim — улучшенный vi. Редактор vi имеет модальный интерфейс — одни и те же клавиши в разных режимах работы выполняют разные действия. В редакторе vi есть три основных режима: командный режим, режим вставки и режим строчного редактора. По умолчанию, работа начинается в командном режиме.

Работа с vi/vim на начальном этапе нетривиальна и по сравнению с другими текстовыми редакторами требует некоторой подготовки. Однако, данный редактор является универсальным для всех операционных систем, поэтому умение работать с vi означает умение настраивать операционную систему даже в случае существенных ограничений: отсутствие графического входа в систему, восстановление после сбоев, удаленная работа.

Редактор vi присутствует только в минимальной установке операционной системы. Во всех других случаях установки в системе используется редактор vim, при этом редактор vi заменяется редактором vim. Главные отличия редактора vim от vi:

– неограниченное число шагов отмены/повторов;
– многооконность;
– поддержка множества буферов;
– подсветка синтаксиса;
– редактирование командной строкой;
– достраивание имен файлов;
– визуальное редактирование;
– графический интерфейс;
– режим совместимости с vi;
– проверка орфографии.

4.8.1. Запуск редактора

В операционной системе всегда присутствует редактор vi, в случае полной установки устанавливается улучшенная версия редактора — vim, в которой присутствуют такие дополнительные возможности как подсветка синтаксиса.

Запустить редактор можно следующим образом:

```
$ vi <файл>
```

или:

```
$ vi [<параметры>] [<список файлов>]
```

Если указано несколько файлов, `vim` по окончанию редактирования первого файла переходит ко второму, после второго — к третьему и так далее.

Улучшенная версия редактора поддерживает навигацию по файловой системе, для этого нужно запустить vi, указав в качестве аргумента каталог — точку входа, например, текущий каталог:

```
$ vim <каталог>
$ vim .
```

Редактор vim является единственным редактором, который поддерживает подсветку синтаксиса практически для всех форматов файлов: исходные тексты программ, различные конфигурационные файлы, файлы журналов. Поэтому его удобно использовать для чтения файлов журналов, а также редактирования конфигурационных файлов. Для чтения файлов журналов рекомендуется использовать режим чтения, для этого нужно вызвать редактор с параметром `-R`:

```
# vim -R <файл журнала>
```

Для чтения большинства журналов необходимы права администратора. Чтение журналов можно выполнять и в режиме редактирования (без параметра `-R`), однако, во избежание случайной порчи файла, рекомендуется всегда использовать `-R`, если файл открывается для чтения.

Примеры:

```
# vim /etc/passwd
# vim -R /var/log/messages
```

В зависимости от типа файла, который `vim` определяет автоматически, меняется подсветка синтаксиса.

4.8.2. Режимы работы универсального текстового редактора `vi`

Режимы работы универсального текстового редактора `vi` с применением клавиатуры:

1) командный — в этом режиме выполняется перемещение курсора, выполнение некоторых действий, переход в другие режимы. В этом режиме редактор работает по умолчанию. Все другие режимы должны активироваться из команд-

ного режима, вернуться обратно в командный режим из любого другого режима — клавиша [Esc];

2) редактирование — в этом режиме введенные символы клавиатуры будут вставляться в текст редактируемого файла, этот режим ещё называют режимом вставки. Обычно переход в режим редактирования осуществляется при помощи клавиши [i];

3) режим строчного редактора `ed` — используется для управления файлами. Переход в данный режим: клавиша [:];

4) режим поиска — ввод поискового запроса. Переход в режим поиска осуществляется клавишей [/];

5) визуальный режим — режим выделения текста, клавиша [v].

4.8.2.1. Командный режим

В командном режиме редактор работает по умолчанию. Переключение в режим редактирования происходит при помощи одной из клавиш, перечисленных в 4.8.2.3. Для наглядности все служебные символы опущены.

Основные действия в командном режиме:

– `^G` — показать сведения о текущем файле и строке.

Запись и выход:

– `ZZ` — сохранить изменения и выйти;

– `ZQ` — выйти без сохранения изменений.

Запись и выход из редактора также возможны через режим редактора `ex` (см. 4.8.2.4).

Перемещение курсора ввода в командном режиме (символ «^» здесь и далее по тексту обозначает клавишу [Ctrl]):

– `вверх`, `вниз`, `вправо`, `влево`, `h`, `j`, `k`, `l` — перемещение курсора ввода клавишами клавиатуры;

– `^U` — табуляция вверх;

– `^D` — табуляция вниз;

– `^F` — на страницу вниз;

– `^B` — на страницу вверх;

– `^E` — подвинуться на строку вниз;

– `^Y` — подвинуться на строку вверх;

– `^` — в начало текущей строки;

– 0 — в начало текущей строки (к первому непробельному символу);

– $ — в конец текущей строки;

– w — на слово вправо;

– b — на слово влево;

– e — в конец слова;

– \\ — на параграф вперед (параграф — это блок текста, отделенный пустой строкой);

– } — на абзац вниз;

– { — на абзац вверх;

– ^f — на страницу (экран) вниз;

– ^b — на страницу (экран) верх;

– [[или]] — в начало текста/в конец текста, при редактировании исходного текста программ перемещается в начало функции/в конец функции;

– ″ — к месту выполнения команды [[;

– % — найти парную скобку (, { или [;

– [n]G — на строку «n»;

– [n]| — в колонку «n»;

– H — в начало экрана;

– M — в середину экрана;

– L — в конец экрана;

– z^ — текущую строку в начало экрана;

– z. — текущую строку в середину экрана;

– z- — текущую строку в конец экрана;

– /<текст> — найти <текст> и перейти к нему (переход в режим поиска);

– / — повторить поиск далее по файлу и перейти к найденному тексту.

Правка в командном режиме:

– x — уничтожить символ после курсора;

– X — уничтожить символ перед курсором;

– d — используется совместно с командами перемещения. Удаляет символы с текущего положения курсора до положения после ввода команды перемещения. Пример: dw — удаляет символы с текущего до конца слова; diw — удаляет слово под курсором;

– c — команда аналогичная d, но после удаление переходит в режим ввода. Очень удобная альтернатива команде R;

– `dd` — стирание текущей строки;
– `J` — слияние текущей строки со следующей;
– `u` — отмена последней команды;
– `U` — отмена всех последних изменений в строке;
– `^r` или `:redo` — вперед по изменениям;
– `.` — повтор последней команды.

4.8.2.2. Повторители

Командам и движениям курсора даются повторители (числа), например:
– `2w` — передвинуть курсор на два слова вперед;
– `10l` — передвинуть курсор на десять символов вправо;
– `d10l` — стереть десять символов справа от курсора;
– `2d10l` — стереть двадцать символов справа от курсора;
– `5J` — слить пять последующих строк в одну;
– `4.` — повторить последнюю введенную команду четыре раза.

4.8.2.3. Режим редактирования

Переход в режим редактирования:
– `i` — вставлять текст в позицию курсора;
– `a` — вставить текст перед позицией курсора;
– `A` — вставлять в конец строки;
– `I` — вставить перед началом строки;
– `R` — перейти в режим замены;
– `r` — заменить один символ;
– `s` — заменить один символ на несколько;
– `O` — вставить строку перед курсором;
– `o` — вставить строку после курсора;
– `C$` — заменить конец строки.
В режиме редактирования:
– `ESC` — возврат из режима редактирования в командный режим;
– `^W`/`^U` — Убрать слово/Убрать строку;
– `Backspace` — удалить символ перед курсором.

4.8.2.4. Режим строчного редактора `ex`

Работа в данном режиме происходит следующим образом: набирается символ «:», вводится необходимая команда, после чего нажимается клавиша [Enter]. Редактор выполняет команду и переходит в обычный режим.

Некоторые команды режима редактора `ex`:

– `:q` или `:q!` — выход из редактора без сохранения изменений;

– `:x` — выход из редактора с записью, если файл был модифицирован;

– `:w` — запись файла;

– `:w <имя файла>` — запись файла;

– `:w! <имя файла>` — запись файла;

– `:e <имя файла>` — загрузка файла `<имя файла>`;

– `:e! <имя файла>` — загрузка файла `<имя файла>`;

– `:r <имя файла>` — добавить содержимое указанного файла к редактируемому сразу за текущей строкой;

– `:set nu` — включить нумерацию строк;

– `:set nonu` — отключить нумерацию строк;

– `:!command` — выполнить команду, как в командной оболочке, не покидая редактора.

4.8.2.5. Режим поиска

Режим поиска позволяет быстро находить текст в документе. Для перехода в режим поиска нужно нажать клавишу [/], ввести текст, который нужно найти, ввод завершить нажатием клавиши [Enter]. Редактор выполнит поиск текста в документе и вернётся в командный режим.

– `/word` — выполнить поиск слова `word` в тексте;

– `/` — повторить поиск слова `word` далее по тексту.

4.8.3. Буферы `vi`

Редактор имеет три типа буферов: буфер стирания (0–9), неименованный буфер, именованные буферы (a-z).

В буферы стирания автоматически заносятся стираемые элементы. В буфере 0 хранится последний стертый элемент, в буфере 1 — предпоследний и т. д.

Занести в буфер:

– yy — занести текущую строку в неименованный буфер;

– y <движение курсора> — занести указанный движением курсора блок текста в неименованный буфер;

– "ayy — занести текущую строку в именованный буфер a;

– "Ayy — добавить текущую строку к содержимому именованного буфера a;

– "by10j — занести последующие 10 строк в именованный буфер b.

Вставить из буфера:

– p — вставить в текущую позицию содержимое неименованного буфера;

– "ap — вставить в текущую позицию содержимое именованного буфера a;

– "1p — вставить в текущую позицию содержимое буфера стирания 1.

4.8.4. Многооконное редактирование

Редактирование нескольких файлов выполняется либо с использованием команды редактора :e <имя файла>, либо указав все необходимые файлы в командной строке при вызове редактора, например, vi file1 file2 file3. В последнем случае передвижение по списку файлов выполняется с помощью команд:

– :n — переходим к следующему файлу в списке;

– :rew — возвращаемся к редактированию первого файла в списке.

Именованные буферы сохраняют свое содержимое при переходе к редактированию другого файла.

5. УСТАНОВКА И УДАЛЕНИЕ ПАКЕТОВ

Операционная система состоит из ядра, набора системных утилит и пользовательских программ. Физически все эти компоненты представляют собой файлы на жёстком диске. Таким образом, добавление новых программ и функциональных возможностей в операционную систему сводится к копированию необходимых для этого файлов: исполняемых, конфигурационных, ресурсов. Удаление программы соответственно сводится к удалению тех же самых файлов. Чтобы упростить процедуру установки, удаления и обновления ПО, в системе установлены пакетные менеджеры: они контролируют, какие файлы нужно добавить или удалить.

ВНИМАНИЕ! НИКОГДА НЕ УСТАНАВЛИВАЕТЕ В ОПЕРАЦИОННУЮ СИСТЕМУ ПРИЛОЖЕНИЯ ПУТЁМ ПРОСТОГО КОПИРОВАНИЯ ФАЙЛОВ. ЭТО МОЖЕТ НАВСЕГДА ПОВРЕДИТЬ ЕЁ. ИСПОЛЬЗУЙТЕ RPM-ПАКЕТЫ И МЕНЕДЖЕР ПАКЕТОВ `YUM` ДЛЯ УСТАНОВКИ ПРОГРАММ.

Менеджер управления пакетами `yum` представляет собой оболочку для `rpm`, обеспечивающую работу с репозиториями с автоматическим разрешением зависимостей. В качестве источников пакетов утилита `yum` использует репозитории, настроить которые системный администратор может при помощи конфигурационных файлов, располагающихся в каталоге `/etc/yum.repos.d`. Типы репозиториев:

– локальный репозиторий — располагаются на локальном запоминающем устройстве или диске (CD, DVD);

– внешний репозиторий — располагаются в сети.

Для работы с `yum` необходимо, чтобы были настроены один или несколько репозиториев, в ОС настроен локальный репозиторий (`media.repo`), который по умолчанию и находятся в каталоге `/etc/yum.repos.d`. Все необходимые программы для ОС находятся на диске дистрибутива. Для доступа к локальному репозиторию необходимо вставить дистрибутив с ОС в привод DVD. Можно подключать другие репозитории для установки сертифицированного ПО, отсутствующего в ОС (см. 5.5).

П р и м е ч а н и е. Для подключения (настройки) репозиториев необходимы права системного администратора.

5.1. yum

Утилита yum позволяет выполнять основные операции по установке и удалению программ, которые оформлены в виде специальных архивов, называемых пакетами. Отличительными особенностями данной программы являются:

– возможность разрешения зависимостей;

– поддержка репозиториев;

– автоматическая работа с несколькими источниками пакетов одновременно.

Формат команды:

```
yum [<команда>] [<параметры>] [<пакет(ы)>]
```

Здесь <команда> может быть одной из нижеследующих:

- install <пакет 1> [<пакет 2>] [...]
- update [<пакет 1>] [<пакет 2>] [...]
- check-update
- upgrade [<пакет 1>] [<пакет 2>] [...]
- distribution-synchronization [<пакет 1>] [<пакет 2>] [...]
- remove | erase <пакет 1> [<пакет 2>] [...]
- list [...]
- info [...]
- provides | whatprovides <возможность 1> [<возможность 2>] [...]
- clean [packages | metadata | expire-cache | rpmdb | plugins | all]
- makecache
- groupinstall <группа 1> [<группа 2>] [...]
- groupupdate <группа 1> [<группа 2>] [...]
- grouplist [hidden] [groupwildcard] [...]
- groupremove <группа 1> [<группа 2>] [...]
- groupinfo <группа 1> [...]
- search <строка 1> [<строка 2>] [...]
- shell [<файл>]
- resolvedep <зависимость 1> [<зависимость 12] [...]
- localinstall <файл rpm 1> [<файл rpm 2>] [...]
- localupdate <файл rpm 1> [<файл rpm 2>] [...]
- reinstall <пакет 1> [<пакет 2>] [...]

— downgrade <пакет 1> [<пакет 2>] [...]

— deplist <пакет 1> [<пакет 2>] [...]

— repolist [all | enabled | disabled]

— version [all | installed | available | group-* | nogroups* | grouplist | groupinfo]

— history [info | list | packages-list | summary | redo | undo | new | addon-info]

— check

— help [<команда>] — справочная информация по команде

Все необходимые программы для ОС находятся на диске дистрибутива. Для установки нужного пакета необходимо вставить дистрибутив с ОС в привод DVD и выполнить следующую команду от имени системного администратора:

```
# yum install <имя пакета>
```

Отдельный файл rpm-пакета устанавливается в систему с помощью команды:

```
# yum localinstall <имя файла.rpm>
```

Программа-установщик по возможности определит и разрешит все необходимые зависимости, а в случае неудачи выдаст список отсутствующих зависимостей.

Обновление системы ОС производится командой:

```
# yum update
```

Для обновления системы ОС необходимо иметь в приводе DVD диск с обновлениями или подключиться к серверу с доступными пакетами обновления.

Для того чтобы обновить только конкретный пакет, нужно выполнить команду:

```
# yum update <имя пакета>
```

5.2. Создание и использование репозитория для yum

Дополнительные репозитории можно подключить, создав файл /etc/yum.repos.d/<имя репозитория>.repo внести в него следующие обязательные строки:

```
[<имя репозитория>]
name=<описание репозитория>
baseurl=<путь до репозитория>
enabled=1
```

gpgcheck=<проверка подписи>

Имя репозитория — короткое имя, по которому к нему будет обращаться администратор, например, myrepo.

Описание репозитория — подробное описание, которое будет выводиться на экран, например «Дополнительное ПО».

Путь до репозитория — один из форматов:

– http://<сетевой адрес>

– ftp://<сетевой адрес>

– file://<полный путь до каталога>

Проверка подписи — проверять цифровую подпись пакетов (gpgcheck=1) или нет (gpgcheck=0). Если источник доверенный, проверку можно отключить.

Проверка доступности подключённого репозитория:

```
# yum clean all
# yum repolist
```

Основные операции yum:

– поиск пакетов в репозиториях:

```
# yum search <имя_пакета>
```

– информация о пакете:

```
# yum info <имя_пакета>
```

– установка пакетов из репозитория:

```
# yum install <имя_пакета>
```

– установка пакетов из файловой системы:

```
# yum localinstall <имя_пакета>
```

– обновление системы:

```
# yum update
```

– удаление пакетов:

```
# yum remove <имя_пакета>
```

– очистка кэша /var/cache/yum:

```
# yum clean all
```

– список доступных пакетов:

```
# yum list <имя_пакета>
```

– список настроенных репозиториев:

```
# yum repolist
```

5.3. rpm

Утилита rpm предназначена для управления пакетами и может использоваться для выполнения различных операций над ними: поиск установленных пакетов, просмотр информации о пакетах, установка, обновление и удаление отдельных пакетов. Программа может использоваться для быстрой установки и обновления одиночных пакетов, но не годится для пакетов со множеством зависимостей, так как не выполняет разрешение зависимостей. В таком случае лучше воспользоваться программой yum.

Формат команды.

Запрос и проверка пакетов:

```
rpm -q|--query [<параметры выбора>] [<параметры запроса>]
rpm -V|--verify [<параметры выбора>] [<параметры проверки>]
rpm --import <публичный ключ>
rpm -K|--checksig [--nosignature] [--nodigest] <файл...>
```

Установка, обновление и удаление пакетов:

```
rpm -i|--install [<параметры установки>] <файл...>
rpm -U|--upgrade [<параметры установки>] <файл...>
rpm -F|--freshen [<параметры установки>] <файл...>
rpm -e|--erase [--allmatches] [--nodeps] [--noscripts]
[--notriggers] [--test] <файл...>
```

Примеры.

Выводит список всех пакетов в системе:

```
rpm -qa
```

Поиск пакета среди установленных в системе.

```
rpm -qa | grep '^coreutils'
```

Выводит информацию о пакете. Используйте [Tab] для автодополнения имени пакета.

```
rpm -qi grep
```

Выводит список файлов, принадлежащих пакету.

```
rpm -ql grep
```

Выводит имя пакета, которому принадлежит данный файл.

```
rpm -qf /bin/pwd
```

Для получения подробной информации наберите команду man rpm.

5.4. Удаление пакетов-сирот

В процессе установки и удаления программ с помощью команды `yum`, может сформироваться набор пакетов, установленных как зависимости, но на текущий момент от этих пакетов ничего не зависит.

Пакеты-сироты образуются следующим образом:

1) устанавливается пакет `A`, который требует для своей работы пакеты `B` и `C`, которые также устанавливаются как зависимости пакета `A`;

2) пакет `A` удаляется (например за ненадобностью или он был установлен по ошибке). При этом `yum` не удаляет зависимости `B` и `C`;

3) если от пакетов `B` и `C` больше не зависит ни один пакет в системе и они не являются самостоятельными приложениями, то пакеты `B` и `C` становятся пакетами-сиротами.

Пакеты-сироты автоматически не удаляются, но системный администратор может периодически отслеживать их.

Для просмотра списка пакетов-сирот выполните команду:

```
$ package-cleanup --quiet --leaves --exclude-bin
```

Параметр `--exclude-bin` сохранит пакеты, в которых есть полезные программы (выводит только служебные пакеты). Если этот параметр опустить, будут также выведены программы, которые могут выполнять полезные функции, например, офисные программы, хотя от этих пакетов ничего не зависит.

Для того чтобы удалить эти пакеты, выполните следующую команду от имени системного администратора:

```
# yum remove $(package-cleanup --quiet --leaves
--exclude-bin)
```

5.5. Управление репозиториями

Репозитории представляют собой текстовые конфигурационные файлы и находятся в каталоге `/etc/yum.repos.d`. Утилиты `yum` считывает все файлы, расположенные в этом каталоге и составляет из них единый список репозиториев, которые будут использоваться при операциях над пакетами. Системный администратор может добавить репозиторий несколькими способами:

– вручную создать файл в каталоге `/etc/yum.repos.d` и заполнить его в соответствии с параметрами репозитория;

– добавить репозиторий при помощи утилиты `yum-config-manager`.

Расширением по умолчанию для конфигурационных файлов репозиториев является `.repo`.

5.5.1. Конфигурационный файл репозитория

Репозитории сохраняются в конфигурационных файлах стандартного формата.

`[<имя>]` Обозначает раздел, в контексте репозиториев обозначает начало нового репозитория с указанным именем. В одном файле можно перечислить несколько репозиториев, но для удобства рекомендуется разделять репозитории на несколько конфигурационных файлов.

`параметр=значение` Стандартная пара параметра и значения.

Рассмотрим основные параметры репозитория.

`name` Произвольное полное название репозитория для удобства администратора.

`baseurl` Адрес репозитория. Может быть файлом (формат `file://<путь>`), или удалённым адресом FTP/HTTP (формат `ftp://<адрес>` или `http://<адрес>` соответственно). Параметр обязателен.

`enabled` Определяет, включён ли репозиторий. Если репозиторий выключен, `yum` будет игнорировать его. Допустимые значения: 0 (выключен) или 1 (включён). Параметр необязателен и по умолчанию репозиторий считается включённым.

`gpgcheck` Проверять электронную цифровую подпись пакета. Допустимые значения: 0 (проверка выключена) или 1 (проверка включена).

Для того чтобы добавить в систему новый репозиторий создайте файл `/etc/yum.repos.d/<имя>.repo` следующего содержания:

```
[<имя>]
name=<полное название репозитория>
baseurl=<адрес репозитория>
gpgckeck=0
```

5.5.2. Создание репозитория с помощью команды `yum-config-manager`

Для того чтобы создать новый репозиторий выполните команду:

```
# yum-config-manager --add-repo <адрес репозитория>
```

Команда автоматически создаст необходимый файл и заполнит недостающие поля.

Для того чтобы включить репозиторий выполните команду

```
# yum-config-manager --enable <имя>
```

Вместо имени можно использовать шаблон. Например:

```
# yum-config-manager --enable repo*
```

Все репозитории, имя которых начинается с «repo» будут включены.

Для того чтобы выключить репозиторий выполните команду

```
# yum-config-manager --disable <имя>
```

Аналогично предыдущему примеру в качестве имени можно использовать шаблон.

6. УЧЁТ ПОЛЬЗОВАТЕЛЕЙ

Наиболее простой и быстрый способ создания пользователя с уровнем доступа предоставляет утилита `z-adm`. Для добавления пользователя `user1` выполните команду:

```
# z-adm user add user1
```

Появится приглашение для ввода пароля пользователя — введите пароль нового пользователя и повторите его. Дождитесь окончания работы команды и появления на экране фразы об успешном добавлении пользователя в систему.

Для подробной информации выполните команду:

```
# z-adm user
```

П р и м е ч а н и е. Утилита `z-adm user` представляет собой объединение стандартных утилит `useradd`, `passwd` и `semanage`. При необходимости системный администратор может выполнить все эти действия вручную.

Создание пользователей происходит с помощью программы `useradd`. Для добавления пользователя выполните эту команду, указав в качестве параметра имя нового пользователя:

```
# useradd <имя нового пользователя>
```

Программа создаст домашний каталог пользователя в каталоге `/home` и выполнит регистрацию нового пользователя в системе.

После создания пользователя необходимо задать ему пароль. Для изменения пароля пользователя используется команда `passwd`:

```
# passwd <имя пользователя, которому нужно изменить пароль>
<введите новый пароль пользователя>
<повторите ввод пароля>
```

Для удаления пользователя используйте команду `userdel`:

```
# userdel <имя существующего пользователя>
```

Для изменения параметров уже существующего пользователя используется команды `usermod`.

Изменить фамилию, имя и отчество пользователя, его офис и телефон позволяет команда `chfn`, которая работает в интерактивном режиме.

6.1. Обмен сообщениями

Так как операционная система многопользовательская, в ней предусмотрен обмен сообщениями между пользователями, работающими в системе.

6.1.1. write

Утилита write передаёт сообщение другому пользователю.

Формат команды:

```
write <адресат>
```

Адресат — имя пользователя или имя пользователя в сети в формате <пользователь>@<сервер>. Команда работает в интерактивном режиме, после выполнения команды пользователь вводит сообщение, завершая ввод сочетанием клавиш [Ctrl+D]. Сообщение появится на терминале адресата.

6.1.2. mesg

Утилита mesg предназначена для настройки приёма сообщений, отправленных другими пользователями командой write.

Формат команды:

```
mesg [-yn]
```

Параметр -y разрешает приём сообщений, -n — запрещает. Запуск команды без параметров выводит текущее состояние разрешения о приёме сообщений.

6.1.3. mail

Утилита mail позволяет обмениваться пользователям почтой. Некоторые программы, например, cron, могут отправлять пользователю почтовые сообщения, информирующие об успехе выполнения заданий.

Отправка почты:

```
mail <адресат>
```

Адресат указывается аналогично команде write, однако адресат получит сообщение в свой почтовый ящик, а на терминал выводится лишь уведомление о наличии новых писем. Прочитать сообщение нужно вручную.

После запуска команда mail выводит сообщение:

```
Subject:
```

и переходит в состояние ожидания ввода. Отправитель должен ввести тему сообщения, затем нажать клавишу [Enter] и вводить текст сообщения. Текст сообщения может состоять из нескольких строк и должен заканчиваться нажатием в начале строки комбинации клавиш [Ctrl+D]. После этого `mail` выводит сообщение:

```
Cc:
```

в ответ на которое можно ввести имя пользователя, которому посылается копия сообщения. Если копия посылаться не должна — нажать клавишу [Enter] для продолжения.

Чтение почты:

```
mail
```

Если для пользователя нет почты, то после запуска команда `mail` выводит сообщение:

```
No mail for <имя пользователя>
```

и завершается.

Если почта есть, то `mail` выводит список имеющихся сообщений, в котором указан статус сообщения, отправитель, дата и время и тема сообщения. После этого `mail` печатает символ «&» — приглашение ко вводу внутренних команд `mail`.

Основные внутренние команды:

– `p` — печать текущего сообщения; сразу после запуска текущим является последнее сообщение;

– `<число>` — установка номера текущего сообщения и печать текущего сообщения;

– `+` — установка текущим следующего сообщения и печать текущего сообщения;

– `R` — формирование и посылка ответа на текущее сообщение;

– `d` — отметка текущего сообщения как удаленного; реальное удаление произойдет после выхода;

– `u` — отмена всех отметок об удалении;

– `s` — сохранение текущего сообщения в файле `mailbox`;

– `x` — окончание работы и выход;

– `?` — получение подсказки по внутренним командам.

7. ФАЙЛЫ, КАТАЛОГИ И ФАЙЛОВЫЕ СИСТЕМЫ

7.1. Работа с файлами и каталогами

Операционная система «ОСь» предоставляет утилиты, предназначенные для работы с данными (файлами и каталогами) в файловой системе.

В файловой системе ОС могут храниться различные объекты, каждый из которых выполняет свою функцию.

Список типов файлов, которые могут храниться в файловой системе ОС:

– обычный файл;

– каталог;

– специальный файл устройства;

– именованный канал;

– сокет;

– символическая ссылка.

В следующих разделах назначение каждого из этих типов рассмотрено подробнее.

Основные возможности по работе с файловой системой:

– просмотр списков файлов и каталогов;

– копирование, создание, перемещение, удаление файлов и каталогов;

– создание жёстких ссылок на файлы;

– создание мягких ссылок на файлы и каталоги;

– чтение различных атрибутов файлов;

– чтение данных из файлов и запись данных в файл;

– просмотр свободного места на носителях.

7.1.1. Файлы и их имена

Компьютер — это инструмент для обработки информации, а информация в любой ОС хранится на физических носителях в виде файлов. С точки зрения операционной системы файл представляет собой непрерывный поток (или последовательность) байтов определённой длины.

Файлы играют важнейшую роль при администрировании ОС. Пользовательские данные все хранятся в виде файлов, однако, также при помощи файлов определяются настройки операционной системы, осуществляется доступ к периферийному оборудованию: дискам, принтерам, терминалам, оперативной памяти, сетевым

адаптерам, а также предоставляется доступ к некоторым внутренним структурам ядра операционной системы. Доступ к файлу равноправен доступу к устройствам, то есть, можно обратиться к принтеру или терминалу как к обычному файлу, назначить на него права доступа, удалить его.

Каждому файлу соответствует индексный дескриптор файла, или `inode`. Именно индексный дескриптор содержит необходимую файловой системе информацию о файле, включая информацию о расположении частей файла на носителе, типе файла. Индексные дескрипторы файлов содержатся в специальной таблице индексных дескрипторов, которая создается при создании файловой системы на носителе. Каждый логический и физический диск имеет собственную таблицу индексных дескрипторов. Дескрипторы в этой таблице пронумерованы последовательно, и именно номер дескриптора файла является его истинным именем в системе. Однако для человека такая система имен неудобна. Поэтому файлам даются еще «человеческие» имена, и, помимо этого, файлы группируются в каталоги.

Для того чтобы администратор идентифицировал файл, файлу присваивается определенная последовательность символов, называемая именем. Файл в операционной системе хранится в каком-либо каталоге, и последовательный список каталогов, заходя внутрь которых можно обнаружить файл, называется путем к файлу, а вся последовательность — путь и имя — называется путевым именем файла. Все файлы в операционной системе хранятся в корневом каталоге, который имеет путь «/». Всё файловое пространство объединено в единое дерево, таким образом, к любому файлу или устройству можно обратиться, начиная с корневого каталога и его путь будет начинаться с символа «/».

Имена файлов имеют длину до 255 символов и состоят из любых символов, кроме символа с кодом `0` и символа «/». Максимальная длина пути до файла не должна превышать 4095 символов. В файловых системах нет понятия расширения файла, но расширение файла может хранится как часть имени файла, при этом с точки зрения операционной системы оно не несет никакой информации. Тем не менее, расширение файла используется для того, чтобы пользователь мог узнать, какого рода данные хранятся в файле.

Таким образом, точка в имени файла в файловой системе ОС не играет никакой роли, если она находится в середине имени. Однако, если точка расположена в начале имени файла, то такой файл с точки зрения ОС является скрытым для пользователя и скрытые файлы имеют особое поведение, отличное от обычных

файлов, например, файлы, в начале имени которых стоит точка, по умолчанию не отображаются файловыми менеджерами.

7.1.2. Обычные файлы

Обычный файл — наиболее распространённый объект в операционной системе, который содержит данные в некотором формате. Для операционной системы в файле содержится абстрактная последовательность байтов и только прикладная программа может произвести интерпретацию содержимого файла и обработать его надлежащим образом. Так, например, обычные текстовые файлы могут открываться практически любой программой или командой, предназначенной для работы с текстом: `cat`, `vim` (см. 7.5.1, 4.8).

Примеры создания обычного файла:

```
$ touch file1
$ echo text > file2
$ cat > file3
line1
line2
Ctrl+D
```

7.1.3. Каталоги

Каталог — специальный файл, который хранит в себе все имена находящихся в нём файлов, а также указатели на другую информацию по отношению к хранящимся в каталоге файлам. Таким образом, право на чтение каталога есть право на чтение имён файлов, содержащихся в данном каталоге, но не самого содержимого файлов. Право на запись в каталог имеет только ядро, а логическая операция создания, перемещения и удаления файлов происходит через интерфейс системных вызовов и контролируется ядром.

Файлы группируются в каталоги, которые, в свою очередь, могут быть включены в другие каталоги. В результате получается иерархическая структура каталогов, начинающаяся с корневого каталога. Каждый каталог содержит как отдельные файлы, так и подкаталоги.

Иерархическую структуру каталогов иллюстрируют рисунком «дерева каталогов», в котором каждый каталог изображается узлом «дерева», а файлы — «листьями». В ОС строится единая структура каталогов для всех носителей, и един-

ственный корневой каталог этой структуры обозначается символом «/». В эту единую структуру каталогов можно подключить любое число каталогов, физически расположенных на разных физических носителях, этот процесс называется монтированием.

Имена каталогов строятся по тем же правилам, что и имена файлов. Фактически, каталоги кроме своей структуры ничем не отличаются от обычных файлов.

Полным именем файла (или полным путем к файлу) называется список имен вложенных друг в друга подкаталогов, начинающийся с корневого каталога и оканчивающийся собственно именем файла. При этом имена подкаталогов в этом списке разделяются тем же символом `/`, который служит для обозначения корневого каталога.

В каждый момент времени пользователь работает с одним экземпляром оболочки Bash и оболочка хранит значение так называемого «текущего» каталога, каталога, в котором пользователь сейчас работает. Имеется специальная команда, которая сообщает значение текущего каталога — `pwd` (см. 7.3.7).

Пример:

```
Login: user
Password: ********
$ pwd
/home/user
```

Понятие «текущий каталог» относится не только к терминалу пользователя, но и вообще к каждому процессу в системе, то есть каждый процесс имеет свой рабочий каталог.

Кроме текущего каталога для каждого пользователя системы определен «домашний каталог» — каталог, в котором пользователь имеет полный доступ ко всем данным. В структуре каталогов ОС домашние каталоги пользователей обычно размещаются в каталоге `/home` и имеют имена, совпадающие с именем пользователя. Когда пользователь входит в систему, по умолчанию его текущим каталогом становится домашний каталог.

В ОС при работе с каталогами можно использовать специальные метасимволы, которые иногда очень удобны. К таким метасимволам относятся:

– `.` (точка) — синоним текущего каталога;

– `..` (две точки) — синоним родительского каталога;

– ~ (тильда) — синоним домашнего каталога.

Метасимволы могут использоваться в любых программах и функциях, предназначенных для работы с файлами.

Пример:

```
$ pwd                  // текущий каталог
/home/user1
$ cd .                 // перехода в текущий каталог
$ pwd                  // текущий каталог не изменился
/home/user1
$ cd ..                // переход на уровень выше
$ pwd                  // текущий каталог изменился
/home
$ cd ~                 // переход в домашний каталог
$ pwd                  // текущий каталог - домашний каталог
пользователя
/home/user1
```

Для изменения текущего каталога служит команда `cd`. В качестве параметра необходимо указать полный или относительный путь к тому каталогу, который необходимо сделать текущим. Относительным путём называется перечисление тех каталогов, которые нужно пройти в дереве каталогов, чтобы перейти от текущего каталога к каталогу целевому. Если целевой каталог, который необходимо сделать текущим, расположен ниже текущего в структуре каталогов, то необходимо указать подкаталог текущего каталога, затем подкаталог того каталога и т. д., вплоть до имени целевого каталога. Если же целевой каталог расположен выше в структуре каталогов или вообще на другой ветви дерева, то применяют метасимволы, описанные выше, например:

```
$ pwd
/home/user/documents/work
$ cd ../..
$ pwd
/home/user
```

7.1.4. Файлы устройств

Файлы устройств — специальные файлы, располагающиеся в каталоге `/dev`. Однако, в отличие от обычных файлов, специальные файлы устройств являются указателями на соответствующие драйверы устройств в ядре.

Работа с устройствами через файлы — одна из возможностей ОС. Операционная система «ОСь» позволяет работать с устройством как с обычным файлом, при этом ядро обеспечивает прозрачное взаимодействие с устройством через драйвер устройства.

Устройства в ОС, располагающиеся в каталоге `/dev/` имеют стандартные предварительно заданные права доступа. Права доступа настроены разработчиками операционной системы. Их изменение должно производиться только квалифицированными специалистами, так как неправильная установка прав доступа может привести к утечке важных данных, сделать систему уязвимой или привести её в нерабочее состояние.

Устройства `/dev/*` не являются фактически существующими и записанными на постоянный носитель файлами. Это виртуальная файловая система ядра, которая создаётся в оперативной памяти каждый раз при загрузке ОС. При перезагрузке все изменения, назначенные на виртуальную файловую систему, пропадают, как и любые другие данные, находящиеся в оперативной памяти. Тем не менее, виртуальная файловая система обладает всеми атрибутами настоящей файловой системы, что позволяет назначать устройствам дискреционные метки.

Назначение дискреционных прав доступа и списков прав доступа (ACL) осуществляется командами `chmod`, `chown`, `setfacl`. Для того чтобы дискреционные метки сохранялись и после перезагрузки системы, достаточно команды, изменяющие их, разместить в файле `/etc/rc.local`. Например, если добавить в файл `/etc/rc.local` строку:

```
setfacl -m u:test_user:rwx /dev/random
```

устройству `/dev/random` будет назначаться запись ACL при каждой загрузке операционной системы, избавляя от необходимости делать это каждый раз вручную.

П р и м е ч а н и е. При изменении типов, уровней или других параметров устройств, система может перестать загружаться или корректно функционировать. Изменение прав доступа устройств возможно, но не является штатным и должно выполняться квалифицированным специалистом.

7.1.4.1. Блочные устройства

Файлы блочных устройств — файлы, которые служат интерфейсом к устройствам, обмен данными с которыми происходит большими фрагментами данных — блоками. Ядро операционной системы обеспечивает всё взаимодействие с таким типом устройств, выбирая драйвер, подходящий для текущего оборудования. Примером блочного устройства служит жёсткий диск

7.1.4.2. Символьные устройства

Файлы символьных устройств — файлы, которые служат интерфейсом к устройствам, осуществляющим собственную буферизацию и побайтовую передачу данных. Примером символьного устройства является терминал.

Одно и то же устройство может иметь два интерфейса — и символьный, и блочный. Например, можно получить побайтовый доступ к содержимому жёсткого диска.

7.1.4.3. Некоторые специальные устройства

Список часто используемых системными администраторами устройств приведён в таблице 7.

Таблица 7 – Перечень некоторых специальных файлов устройств

Устройство	Описание
`/dev/random`	Датчик случайных чисел, который преобразует тепловой шум устройств системы в случайные биты данных. Отличается медленной скоростью, но в то же время высокой надежностью (крайне низок шанс предугадать последовательность, полученную из `/dev/random`). Устройство является эмулятором аппаратного генератора случайных чисел
`/dev/urandom`	В отличие от `/dev/random`, генерирует ровно столько данных, сколько было запрошено без задержек. Обладает хорошей скоростью и большим периодом

Устройство	Описание
/dev/null	«Чёрная дыра». Все, что было отправлено в это устройство будет утеряно навсегда. Используется для перенаправления вывода ошибок или стандартного вывода, если он не нужен, полезно для использования в сценариях Bash
/dev/zero	Генератор, аналогичный /dev/urandom, но вместо случайной последовательности, генерирует последовательность нулей. Устройство используется для создания заполненных нулевым байтом файлов заданной длины
/dev/cdrom	Устройство привода чтения компакт-дисков
/dev/ttyN	Терминал
/dev/mem	Физическая оперативная память, память выше 1 Мбайт недоступна
/dev/kmem	Виртуальная память ядра
/dev/sd*	Устройство жёсткого диска, например, /dev/sda. Если на диске несколько логических разделов, им присваиваются номера, например, /dev/sda1 и /dev/sda2 означают соответственно первый и второй логические разделы на диске /dev/sda

7.1.5. Именованные каналы

Каналы — особые файлы, через которые процессы обмениваются данными, одна из форм межпроцессного взаимодействия. Каналы являются аналогами конвейера.

Вместо традиционного, безымянного конвейера (см. 4.2.5), явно создаётся файл с помощью команды mkfifo, к которому могут обратиться несколько различных процессов по имени.

Пример работы с именованным каналом:

```
$ mkfifo test_pipe // создать канал test_pipe
$ cat test_pipe // извлечь информацию из test_pipe
//Данную команду нужно выполнить в другом терминале
$ echo TEST_DATA > test_pipe // Осуществить передачу
информации в test_pipe
//В первом терминале должна появиться строка TEST_DATA
```

7.1.6. Символьные ссылки

Символьная ссылка — специальный файл, в котором хранится указатель на другой файл — относительный или абсолютный путь. Это отличает символические ссылки от жёстких ссылок, которые являются дубликатом `inode`.

Целью ссылки может быть любой объект — другая ссылка, файл, каталог, или даже несуществующий файл. Ссылки используются для удобной организации структуры файлов, кроме того, символьная ссылка может ссылаться на каталоги и на другой дисковый раздел, что невозможно осуществить при помощи жёсткой ссылки. Символьные ссылки создаются командой `ln` с параметром `-s`. Жёсткие ссылки создаются также командой `ln`, но без параметра (см. раздел 7.3.6).

Пример создания ссылок:

```
$ mkdir ~/test_dir
$ cd ~/test_dir
$ pwd
/home/user1/test_dir
$ mkdir dir1
$ ln dir1 dir2
ln: «dir1»: не допускается создавать жёсткие ссылки на
каталоги
$ touch file1
$ ln file1 file2
$ ls -1i              // обратите внимание на одинаковый
inode
14943954 dir1
14943893 file1
14943893 file2
$ ln -s dir1 dir2
$ ls -dil dir1 dir2     // inode для символьных ссылок
разные
14943954 drwxrwxr-x. 2 rpg rpg 4096 Июл  1 12:50 dir1
14943955 lrwxrwxrwx. 1 rpg rpg    4 Июл  1 12:52 dir2 ->
dir1
```

Определить наличие жёсткой ссылки очень сложно. Дело в том, что ни ссылка, ни сам файл не содержат никакой информации о местонахождении второго файла.

Определить наличие жёсткой ссылки на файл возможно только командой `ls -l`, которая во втором столбце отображает количество жёстких ссылок на файл. Если это число больше единицы — это означает, что где-то существует жёсткая ссылка на данный файл (но не сообщает, где именно).

Пример:

```
$ ls -l file1 file2
-rw-rw-r--. 2 rpg rpg 0 Июл  2 09:21 file1
-rw-rw-r--. 2 rpg rpg 0 Июл  2 09:21 file2
            |---количество жёстких ссылок
```

7.1.6.1. Удаление символьных и жёстких ссылок

Удаление символьной ссылки приведёт только у удалению самой ссылки, но не удалению самого файла, на который она ссылается. Удаление файла, на который ссылается символьная ссылка, приведёт к тому, что данная ссылка станет нерабочей («битая ссылка»), при этом данные с жёсткого диска пропадут навсегда.

Удаление жёсткой ссылки не приведёт к удалению самого файла, а только к удалению данной ссылки. Данные при этом сохраняются. Однако, если удалить сам файл, данные также сохранятся и будут доступны по имени ссылки. Особенность жёстких ссылок состоит в том, что невозможно после создания жёсткой ссылки на файл определённо сказать, какой файл является оригинальным, а какой — ссылкой, т. е. жёсткая ссылка и сам файл равноправны. Данные с диска стираются только после того, как на данный файл не останется ни одной жёсткой ссылки.

7.1.7. Сокеты

Сокеты — одна из форм межпроцессного взаимодействия, также могут использоваться для доступа к сети. Многие системные службы работают через сокеты, что обеспечивает сетевую прозрачность.

7.2. Структура файловой системы

Структура файловой системы организована в формате общепринятых имён файлов и каталогов, которые расположены в виде удобной логической структуры. Это облегчает процедуру администрирования, а также упрощает общение системных администраторов, имеющих опыт общения с UNIX-системами.

В данном разделе приведено краткое описание основных каталогов файловой системы.

7.2.1. Корневой каталог /

Корневой каталог является основной входной точкой файловой системы. Все остальные файлы и каталоги располагаются внутри корневого каталога.

7.2.2. /bin

В каталоге /bin находятся основные часто используемые команды. Основное назначение данного каталога — возможность размещения базового каталога с бинарными файлами /usr/bin на другом разделе жёсткого диска, однако такая возможность обычно не используется.

7.2.3. /dev

Каталог /dev содержит файлы устройств, которые представляют собой интерфейс к драйверам периферийных устройств. В /dev находится несколько подкаталогов, в которых группируются файлы одного вида, например, /dev/pts содержит все виртуальные терминалы.

7.2.4. /etc

В данном каталоге расположены все системные конфигурационные файлы. Наиболее важные из них — сценарии инициализации: /etc/rc*. Через данный каталог происходит системное администрирование либо вручную — путём правки конфигурационных файлов, либо автоматически — путём выполнения соответствующих команд.

7.2.5. /home

Стандартный каталог для пользовательских домашних каталогов. Обычно домашний каталог принято создавать как /home/<имя пользователя>.

7.2.6. /lib

Данный каталог содержит основные библиотеки. В нём находятся только базовые из них, остальные библиотеки расположены в каталоге /usr/lib64.

7.2.7. /media

Данный каталог служит для временного монтирования файловых систем таких как USB флеш-устройства и компакт-диски. Монтирование в данный каталог осуществляется автоматически.

7.2.8. /misc

В данный каталог происходит автоматическое монтирование службой automount. В частности, в каталог /misc/cd монтируется любой компакт-диск, если попытаться прочитать содержимое /misc/cd.

7.2.9. /proc

Специальный каталог, предоставляющий доступ к некоторым внутренним структурам ядра через файлы. Все файлы в каталоге /proc создаются «на лету» в момент обращения к ним.

7.2.10. /sbin

В данном каталоге находятся основные программы, используемые для администрирования операционной системы.

7.2.11. /usr

Это самый большой каталог в системе, он содержит исполняемые файлы (/usr/bin), библиотеки (/usr/lib64), ресурсы приложений (/usr/share) и т. п.

7.2.12. /var

Данный каталог предназначен для хранения временных файлов различных служб, например, печати. /var/log — важный каталог, хранящий файлы журналов. По умолчанию для /var/log выделяется отдельный дисковый раздел размером 200 мегабайт.

7.2.13. /tmp

Каталог для хранения временных файлов.

7.2.14. /tmp-noinst

Специальный временный каталог, который не является многоэкземплярным и используется как классический /tmp в UNIX.

7.3. Утилиты для работы с файлами и каталогами

Для манипулирования файлами из командной строки в ОС используются утилиты, перечисленные в данном разделе.

7.3.1. ls

Утилита ls служит для вывода на экран списка имен файлов и подкаталогов текущего каталога.

Если команда ls запущена без параметров, то выводятся только имена файлов текущего каталога. Если нужно просмотреть содержимое другого каталога, надо указать полный или относительный путь к каталогу.

Пример:

```
$ ls /
bin   dev  home   lib  mnt   proc     sbin srv  tmp  var
boot  etc  media  opt  root  sys  usr
```

Кроме того, утилита ls выводит не только список файлов и каталогов, но и их атрибуты. Для того чтобы отобразить список файлов с атрибутами, используется команда ls -l:

```
$ ls -l /
итого 166
drwxr-xr-x   2 root root   3520 Авг  1 13:36 bin
drwxr-xr-x   3 root root   1144 Авг  2 10:31 boot
drwxr-xr-x  18 root root   3720 Авг  3 15:11 dev
drwxr-xr-x 116 root root   8096 Авг  3 16:01 etc
...
```

Формат команды:

```
ls [<параметры>] [<файл...>]
```

Можно вызывать утилиту `ls` с разными параметрами для получения той или иной информации. Список наиболее полезных параметров утилиты `ls`:

– `-a`, `--all` — отображать скрытые файлы, начинающиеся с точки — «.»;

– `-A`, `--almost-all` — выдавать все файлы, кроме «`.`» и «`..`»;

– `-B`, `--ignore-backups` — не показывать файлы, которые заканчиваются на «~», если они не заданы в командной строке;

– `-c`, `--time=ctime`, `--time=status` — сортировать содержимое каталога в соответствии с временем изменения состояния файла (поле `ctime` в `inode`). Если с помощью параметра `-l` задан длинный формат, то выдавать время изменения состояния файла вместо времени его модификации.

– `-d`, `--directory` — выдавать имена каталогов, а не их содержимое, а также не следовать по символьным ссылкам;

– `-f` — не сортировать содержимое каталога; выдавать файлы в том порядке, в котором они записаны на диск;

– `-F`, `--classify`, `--indicator-style=classify` — добавлять к каждому имени файла символ, показывающий его тип. Для обычных исполняемых файлов это «`*`». Для каталога добавляется «`/`», для именованных каналов — «`|`», для символьных ссылок «`@`», для сокетов «=», для обычных файлов ничего не добавляется;

– `-G`, `--no-group` — не отображать информацию о группе в длинном формате вывода;

– `-h`, `--human-readable` — с параметром `-l`, печатать размеры файлов в удобном для человека виде, например, 1K, 234M, 2G;

– `-H`, `--si` — делает то же, что и параметр `-h`, но использует официальные единицы измерения СИ (где для расчётов используется 1000 вместо 1024 и, таким образом, M — это 1000000, а не 10485576);

– `-i`, `--inode` — выдавать номер `inode` каждого файла, слева от его имени. Этот номер однозначно идентифицирует каждый файл в каждой файловой системе (см. раздел 7.1.1);

– `-I`, `--ignore=шаблон` — не показывать файлы, имена которых совпадают с заданным шаблоном, если только они не заданы в командной строке. Как и в Bash, начальная «`.`» в имени файла не совпадает с символом «`*`», заданным в начале шаблона (см. раздел 4.2.8.6);

– `-l` — использовать «широкий формат» — подробная информация по каждому файлу;

– -L, --dereference — выдавать информацию о файлах, на которые указывают символические ссылки, вместо информации о самих символических ссылках;

– -R, --recursive — рекурсивно показывать каталоги;

– -S, --sort=size — производить сортировку по размеру файла, вместо сортировки по алфавиту. Таким образом, наибольшие файлы будут показаны сначала;

– --color[=none|auto|always] — задаёт цвет для различения типов файлов. Цвета задаются с использованием переменной окружения LS_COLORS. none — не использовать цвет, установлено по умолчанию; auto — использовать цвет, только если стандартный вывод является терминалом; always — всегда использовать цвет, --color без параметра эквивалентно --color=always;

– --full-time — выдавать время в полном, а не в стандартном сокращённом варианте;

– -1 — выводить по одному имени файла на строке. Этот параметр включается по умолчанию, если стандартный вывод не является терминалом;

– -Z — отображать контекст безопасности КСЗ, автоматически активирует параметр -l;

– --scontext — вывести только контекст безопасности КСЗ.

Длинный формат вывода команды ls с параметром -l нуждается в дополнительном пояснении. Вывод команды разбивается на колонки:

1) дискреционные права доступа в символьном виде. Первый символ обозначает тип файла:

– - (дефис) — обычный файл;

– d — каталог;

– b — блочное устройство;

– c — символьное устройство;

– l — символическая ссылка;

– p — именованный канал;

– s — сокет;

2) второй столбец для каталогов отображает количество объектов внутри каталога, а для файла — количество жёстких ссылок;

3) имя пользователя владельца файла;

4) имя группы владельца файла;

5) размер файла в байтах. Для каталогов данный столбец отображает не суммарный размер файлов внутри этого каталога, а размер самого каталога, который он занимает на диске;

6) месяц последнего изменения файла;

7) день последнего изменения файла;

8) время последнего изменения файла;

9) имя файла.

Если включена цветовая индикация, цвета файлов означают следующее:

– синий — каталог;

– зелёный — с правом на исполнение;

– жёлтый — файл устройства;

– красный — архив;

– фиолетовый — изображение;

– светло-голубой — мультимедийный файл;

– светло-голубой с символом `->` — ссылка, после стрелки отображается путь, на который указывает ссылка.

7.3.2. `lsattr`

Команда `lsattr` позволяет просмотреть расширенные атрибуты, установленные с помощью команды `chattr`. В ее выходных данных используются те же буквы, что и в команде `chattr`.

Формат команды:

```
lsattr [-Rad] [<файл...>]
```

Параметры аналогичны параметрам команды `ls`.

7.3.3. `mkdir`

Утилита `mkdir` создает новый каталог. Параметр `-p` указывает на необходимость создания полной цепочки каталогов, которых ещё не существует.

Формат команды:

```
mkdir [<параметры>] <каталог...>
```

Параметры:

– `-m`, `--mode=`*режим* — установить код доступа (как в chmod);

– `-p`, `--parents` — не выдавать ошибок, если такой каталог уже существует, создавать родительские каталоги, если необходимо;

– -v, --verbose — печатать сообщение о каждом созданном каталоге;

– -Z, --context=<контекст> — установить контекст безопасности КСЗ для каждого создаваемого каталога равным <контекст>.

Примеры.

Создание каталога newdir:

```
$ mkdir newdir
```

Последовательное создание следующих каталогов (будет создан каталог newdir2/subdir1/subdir2, а также newdir2/subdir1, если он не существует и newdir2, если такой каталог тоже не существует):

– newdir2

– newdir2/subdir1

– newdir2/subdir1/subdir2

```
$ mkdir -p newdir2/subdir1/subdir2
```

Создание каталогов dir1 и dir2:

```
$ mkdir dir{1,2}
```

7.3.4. mknod

Специальные файлы блочных или символьных устройств, а также именованные каналы можно создать при помощи команды mknod.

Формат команды:

```
mknod [<параметры>] <имя> {bc} <основной номер>
<второстепенный номер>
mknod [<параметры>] имя p
```

mknod создает файлы трёх типов: именованный канал, специальный файл символьного или блочного устройства. Специальный файл записывается в файловой системе с помощью трёх параметров: одного логического и двух целых. Логический параметр говорит о том, является ли специальный файл символьным или блочным. Два целых параметра задают основной и второстепенный номера устройства. Таким образом, специальный файл практически не занимает места на диске и используется только для общения с операционной системой, а не для хранения данных. Часто специальные файлы ссылаются на аппаратные устройства (диск, ленточное устройство, терминал, принтер) или на службы операционной системы (/dev/null, /dev/random). Специальные блочные файлы обычно ссылаются на устройства, подобные диску (где данные могут быть получены с помощью номера

блока, и, например, такие устройства может иметь кэш блоков). Все другие устройства представлены специальными символьными файлами. Аргумент, следующий за аргументом имя, задает тип файла, который необходимо создать:

– p — для именованного канала;

– b — для блочного специального файла;

– c — для символьного специального файла.

Когда создается специальный блочный или символьный файл, то после типа файла должны быть указаны основной и второстепенный номера устройства (в десятичной или восьмеричной форме с ведущим нулем). По умолчанию значением прав доступа к созданным файлам становится 0666 (a+rw).

Параметры:

-m права, --mode=права

Значение прав доступа к создаваемым файлам становится равным по величине значению аргумента права; оно может иметь как символьную форму (например, rwx), так и восьмеричную (например, 0777).

--help

Выводит подсказку на стандартный вывод и завершает свою работу.

--version

Выводит информацию о версии программы на стандартный вывод и завершает свою работу.

В отличие от специальных файлов устройств, создаваемых операционной системой в каталоге /dev (который является специальной файловой системой devtmpfs, хранящейся в оперативной памяти), программа mknod позволяет создавать специальные файлы в обычной файловой системе, что обеспечивает сохранение назначенных им дискреционных прав доступа (в том числе списков прав доступа — ACL), создаются вновь при загрузке операционной системы, что обеспечивает гибкость при подключении к компьютеру различных устройств, поскольку операционная система создает специальные файлы только для тех устройств, которые в данный момент подключены к компьютеру).

7.3.5. touch

Утилита touch устанавливает время последнего изменения файла на текущее, если файл не существует — создаёт пустой файл:

```
$ touch newfile
```

```
$ ls
newfile
```

Формат команды:

```
touch [<параметры>] [-r <файл>] [-t <время>] <файл...>
```

Параметры:

– `-a` — устанавливает время последнего доступа к файлу. Время последнего изменения не устанавливается, если явно не задан ключ `-m`;

– `-c` — указывает утилите не создавать файл, если он не существует, при этом никаких сообщений об ошибке показано не будет;

– `-f` — пытается обновить информацию о времени, даже если права доступа файла не позволяют делать;

– `-h` — указывает утилите не изменять данные о файле, если он задан символической ссылкой;

– `-m` — устанавливает время последнего изменения файла;

– `-r <файл>` — использовать значения времени из файла, заданного аргументом `<файл>`;

– `-t <время>` — устанавливает время последнего изменения и доступа в соответствии с указанным форматом `<время>`.

Формат даты, указанный в ключе `-t` задается в соответствии с шаблоном `[[CC]YY]MMDDhhmm[.SS]`:

– `CC` — первые две цифры года;

– `YY` — последние две цифры года, если параметр `CC` не задан и значение `YY` находится в пределах 69 и 99, то тогда `CC` устанавливается равным 19, в противном случае используется 20;

– `MM` — двузначный номер месяца;

– `DD` — двузначный номер дня;

– `hh` — часы;

– `mm` — минуты;

– `SS` — секунды.

7.3.6. `ln`

Команда `ln` позволяет создавать символьные и жёсткие ссылки.

Формат команды:

```
ln [<параметры>] <цель> [<назначение>]
```

или

`ln [<параметры>] <цель...> <каталог>`

Существует два вида ссылок, обычно называемых жёсткие ссылки и символьные, или «мягкие» ссылки. Жёсткая ссылка является всего лишь именем какого-либо файла. Таким образом, файл может иметь несколько имён. Он будет удалён с диска только тогда, когда будет удалено последнее из его имён. Количество имён, которые имеет файл, показывает команда `ls`.

Символьная ссылка отличается от жёсткой ссылки: она является маленьким специальным файлом, который содержит путь к другому файлу. Таким образом, мягкая ссылка может указывать на файлы, которые находятся на других файловых системах (например, смонтированных по NFS с сервера) и не нуждается в наличии того файла, на который она указывает. Когда происходит попытка доступа к символьной ссылке, ядро операционной системы заменяет ссылку на тот путь, который она содержит. При этом нужно помнить, что команда `rm` удаляет саму ссылку, а не файл, на который она указывает.

`ln` создаёт ссылки между файлами. По умолчанию создаются жесткие ссылки; при указании параметра `-s`, создаются символьные ссылки.

Если задан только один файл, то для него создаётся ссылка в текущем каталоге с таким же именем, как у этого файла. В противном случае, если последний аргумент является именем существующего каталога, `ln` создаст ссылки в этом каталоге для каждого из исходных файлов, с такими же именами как и у исходных файлов. В противном случае, если задано два файла, то создается ссылка с заданным именем для исходного файла. Если последний аргумент не является каталогом и задано более чем два аргумента, то будет выдаваться сообщение об ошибке.

По умолчанию `ln` не удаляет существующие файлы или существующие символьные ссылки, но можно принудительно удалять файлы и ссылки, указав параметр `-f`.

Параметры:

– `-f`, `--force` — удалять существующие файлы;

– `-i`, `--interactive` — запрашивать подтверждение удаления файлов;

– `-n`, `--no-dereference` — считать файл назначения обычным файлом, если явно заданный параметр является символьной ссылкой на каталог;

– `-s`, `--symbolic` — делать символьные ссылки вместо жёстких ссылок.

7.3.7. pwd

Утилита pwd отображает текущий рабочий каталог. Текущий рабочий каталог — каталог, относительно которого вычисляются относительные пути файлов. Например, если текущий рабочий каталог — /tmp, то при обращении из командной строки к файлам file1 и dir1/file2 операционная система будет искать файлы, имеющие полный путь /tmp/file1 и /tmp/dir1/file2 соответственно.

7.3.8. cd

Утилита cd устанавливает новый текущий рабочий каталог.

Примеры.

Переход в домашний каталог:

```
$ cd
```

Переход на уровень выше:

```
$ cd ../
```

Для отображения текущем рабочем каталоге используйте команду pwd.

7.3.9. cp

Утилита cp копирует файл, дерева каталогов. Формат команды:

```
$ cp [<параметры>] <источник> <назначение>
```

Возможны следующие параметры:

– -f или --force — принудительная перезапись файлов;

– -i или --interactive — интерактивный режим, программа спрашивает разрешения на перезапись;

– -n или --no-clobber — не перезаписывать существующий файл;

– -R или -r или --recursive — рекурсивно копировать каталоги.

Пример использования:

```
$ cp file file_copy
$ cp *.txt ./texts/
$ cp -r dir newdir
```

7.3.10. mv

Утилита mv перемещает или переименовывает файлы.

Формат команды для переименования файла:

```
$ mv [<параметры>] <имя файла> <новое имя>
```

Формат команды для перемещения файла с именем <имя файла> в <каталог>.

```
$ mv [<параметры>] <имя файла> <каталог>
```

Формат команды для перемещения файла по пути:

```
$ mv [<параметры>] <путь до файла> <путь>
```

Параметры:

– -f, --force — удаляет <новое имя>, если файл существует, не спрашивая об этом пользователя;

– -i, --interactive — просит подтверждения на замену существующего файла, в виде вопроса, которые выводится на стандартный вывод ошибок и читает ответ из стандартного ввода. Если ответ не утвердительный, то файл пропускается;

– -u, --update — не переносит не-каталоги, которые уже существуют в месте, куда осуществляется перенос и имеют то же самое или более позднее время модификации.

Примеры.

Переименование файла:

```
$ mv old_name new_name
```

Перемещение файла в каталог /tmp:

```
$ mv file /tmp
```

Перемещение всех текстовых файлов в каталог texts:

```
$ mv *.txt ./texts
```

Перемещение файла из каталога /tmp в /var/tmp:

```
$ mv /tmp/file_name /var/tmp
```

Параметры данной команды аналогичны параметрам утилиты cp за исключением, что невозможно использовать параметр рекурсии -r.

7.3.11. rm

Утилита rm предназначена для удаления файлов и каталогов. Файлы удаляются безвозвратно, без предупреждения, поэтому её нужно использовать с осторожностью.

Формат команды:

```
$ rm [<параметры>] <имя файла>
```

Параметры данной утилиты аналогичны параметрам утилиты `cp` (см. 7.3.9). Необходимо соблюдать осторожность при использовании символов метаподстановки или рекурсии: это может привести к случайному удалению важных файлов. Рекомендуется использовать утилиту в интерактивном режиме.

7.3.12. `rmdir`

Утилита `rmdir` предназначена для удаления пустых каталогов. Команда не может удалить каталог, если он не пуст, для этих целей используйте команду `rm -r`.

Формат команды:

```
$ rmdir <каталог>
```

Параметры данной утилиты аналогичны параметрам утилиты `cp`. Необходимо соблюдать осторожность при использовании символов метаподстановки или рекурсии: это может привести к случайному удалению важных файлов. Рекомендуется использовать утилиты в интерактивном режиме.

П р и м е ч а н и е. При необходимости удаления каталога, внутри которого есть файлы, нужно использовать команду `rm -r`.

7.3.13. `mkfifo`

Утилита `mkfifo` используется для создания специальных файлов: именованных каналов. С помощью этих каналов можно осуществлять взаимодействие между процессами операционной системы.

Именованный канал аналогичен неименованному (который устанавливается с помощью символа `|`) — процессы могут писать в него и читать из него. При этом с каналом оперируют как с файлом:

```
$ echo ”Hello, world.” > pipe
$ cat < pipe
Hello, world.
```

Синтаксис команды:

```
$ mkfifo <имя именованного канала>
```

Канал удаляется или перемещается как обычный файл.

Каналы, в отличие от файлов, только передают, но не сохраняют информацию. Например, последующее чтение канала приведёт к приостановке просматривающего процесса: пока не поступят новые данные.

Именованные каналы — одна из форм межпроцессного взаимодействия.

7.3.14. `find`

Утилита `find` используется для поиска файлов.

Числовые параметры могут задаваться следующим образом:

– `+n` — больше, чем `n`;

– `-n` — меньше, чем `n`;

– `n` — точно `n`.

Критерии поиска:

– `amin n` — доступ к файлу осуществлялся `n` минут назад;

– `cmin n` — статус файла изменялся `n` минут назад;

– `empty` — пустой файл;

– `executable` — исполняемый файл;

– `iregex pattern` — поиск файла по регулярному выражению (подробное описание регулярных выражений приведено в разделе 7.6);

– `mmin n` — файл был изменен `n` минут назад;

– `name <шаблон>` — поиск файлов, имя которых соответствует шаблону;

– `size n[b|c|w|k|M|g]` — размер файла, можно указать необязательное окончание:

  – `b` — блоки по 521 байт (по умолчанию);

  – `c` — байт;

  – `w` — два байта;

  – `k` — килобайт;

  – `M` — мегабайт;

  – `G` — гигабайт;

– `type c` — файл определённого типа:

  – `b` — блок;

  – `c` — символ;

  – `d` — каталог;

  – `p` — именованный канал;

  – `f` — обычный файл;

  – `l` — сивольная ссылка;

  – `s` — сокет.

Эти параметры могут использоваться вместе.

Пример:

```
$ find -name ”*.txt”
```

К найденным файлам могут применяться следующие действия:

– `-print` — вывести на экран полный путь до найденного файла. Это действие выполняется по умолчанию;

– `-print` — аналогично предыдущему, но в качестве разделителя выступает нулевой байт (полезно для некоторых программ и не предназначено для чтения человеком);

– `-printf <шаблон>` — форматировать вывод в соответствии с `<шаблоном>`;

– `-delete` — удалить найденные файлы;

– `-exec <команда> {} \;` — исполнение указанной команды для каждого найденного файла с передачей имени файла в качестве аргумента (вместо спецсимвола `{}` выполняется подстановка имени файла);

– `-ok <команда> {} \;` — аналогична действию `-exec`, только для каждого файла запрашивается подтверждение перед выполнением команды.

П р и м е ч а н и е. В действии `-exec` и `-ok` важно соблюдать всю последовательность символов, включая пробелы и знак экранирования точки с запятой (`\;`).

Примеры.

Вывод всех файлов в текущем каталоге и подкаталогах:

```
$ find
.
./xxx
./xxx/yyy
./xxx/yyy/zzz
./test.file
```

Вывод всех файлов из каталога `/etc`, начинающихся на `re`:

```
$ find /etc -name ”re*”
/etc/apm/resume.d
/etc/ppp/resolv.conf
/etc/pam.d/rexec
/etc/ssmtp/revaliases
/etc/conf.d/reslisa
/etc/init.d/reslisa
```

```
/etc/init.d/reboot.sh
/etc/resolv.conf
```

Вывод всех файлов из каталога `/etc`, изменённых за последние сутки:

```
$ find /etc -mtime -1
/etc
/etc/mtab
/etc/env.d/01hostname
/etc/adjtime
```

Вывод всех файлов из каталога `/tmp`, не принадлежащих пользователю `user`:

```
$ find /tmp \! -user user
/tmp/.X11-unix
/tmp/.X11-unix/X0
/tmp/mc-root
```

Так как символ «!» имеет особое значение в командной оболочке, его нужно экранировать.

Удаление в текущем каталоге всех файлов, оканчивающихся тильдой (~):

```
$ find -name "*~" -delete
```

Копирование всех текстовых файлов на USB-накопитель:

```
$ find -name "*.txt" -exec cp  /media/disk/ \;
```

7.3.15. `which`

Команда `which` отображает полный путь к указанным командам или сценариям. Для каждого из своих аргументов она выводит тот полный путь к исполняемому файлу, который будет использован командной оболочкой, если имя программы ввести в качестве команды в командной строке. Эта программа выполняет поиск исполняемых файлов или сценариев в каталогах, перечисленных в переменной окружения `PATH`, используя тот же алгоритм, что и командная оболочка Bash.

Формат команды:

```
which [<параметры>] <команда...>
```

Параметры:

– `--all`, `-a` — выводит все совпавшие исполняемые файлы по содержимому в переменной окружения `PATH`, а не только первый из них;

– `--read-alias`, `-i` — считывает псевдонимы, поступающие из стандартного ввода и направляет на стандартный вывод информацию по совпавшим. Этот

параметр полезна в сочетании с использованием псевдонима для самой команды `which`. Например: `alias which='alias | which -i'`;

– `--skip-alias` — игнорирует параметр `--read-alias`, если таковая имеется. Этот параметр полезна для точного поиска обычных двоичных файлов, которые используют параметр `--read-alias` в псевдониме или функции для `which`;

– `--read-functions` — считывает функции, определённые в командной оболочке и поступающие из стандартного ввода, а затем направляет на стандартный вывод информацию по совпавшим. Этот параметр полезен в сочетании с функциями командной оболочки для самой команды `which`. Например:

```
which() { declare -f | which --read-functions $@ }
export -f which
```

– `--skip-functions` — игнорирует параметр `--read-functions`, если таковая имеется. Полезно для точного поиска обычных двоичных файлов, которые используют параметр `--read-alias` в псевдониме или функции для `which`;

– `--skip-dot` — пропускает все каталоги из переменной окружения `PATH`, которые начинаются с точки;

– `--skip-tilde` — пропускает все каталоги из переменной окружения `PATH`, имена которых начинаются с символа тильда «~», а также все исполняемые файлы, из `$HOME`;

– `--show-dot` — если имя каталога из переменной окружения `PATH` начинается с точки и соответствующий исполняемый файл `имя_программы` был найден в этом пути, тогда вместо полного пути будет выведено `./имя_программы`;

– `--show-tilde` — выводит тильду, когда каталог совпадает с каталогом, указанном в переменной окружения `HOME` (то есть с домашним каталогом). Этот параметр игнорируется, если `which` вызывается системным администратором;

– `--tty-only` — не обрабатывает параметры, которые находятся справа, если они поступают не с терминала (`tty`).

7.3.16. `type`

Команда `type`, как и `which`, осуществляет поиск исполняемых файлов в пути поиска вашего командного процессора. Однако команда `type` является встроенной в командную оболочку и работает быстрее, тогда как `which` — это программа на диске.

Формат команды:

```
type <имя программы...>
```

Отличием `type` от `which` является возможность определения синонимов и встроенных в командную оболочку команд, например:

```
$ type pwd
pwd is a shell builtin
$ type ls
ls is aliased to 'ls --color=auto'
$ type cat
cat is /bin/cat
```

7.3.17. `whereis`

`whereis` — утилита, которая выводит информацию о расположении файлов определенного приложения.

Формат команды:

```
whereis [<параметры>] <файл>
```

Параметры:

– `-b` — искать только исполняемые файлы;

– `-B <каталог...>` — указывает, в каком каталоге производить поиск;

– `-f` — очистить список каталогов поиска и использовать только те, что указаны явно параметрами;

– `-m` — искать только разделы руководства;

– `-M <каталог...>` — указывает, в каком каталоге производить поиск руководств;

– `-s` — искать только исходные тексты;

– `-S <каталог...>` — указывает, в каком каталоге производить поиск исходных текстов;

– `-u` — искать все файлы, не попадающие ни в одну из категорий (исполняемые файлы, файлы руководств, исходные тексты).

7.3.18. `stat`

Команда `stat` выводит содержимое полей дескриптора файла или статус файловой системы.

Формат команды:

```
stat [<параметры>] <файл...>
```

Параметры:

– `-c, --format=<формат>` — применяется указанный формат вывода `<формат>` (см. далее), вместо используемого по умолчанию;

– `-f, --filesystem` — выводит статус файловой системы (на которой расположен `FILE`) вместо статуса файла;

– `-L, --dereference` — выводит информацию о статусе оригинального файла, с которым связана ссылка `<файл...>`;

– `-t, --terse` — выводит информацию в сокращённой (сжатой) форме.

В качестве формата вывода `<формат>` для файлов (но не для параметра `-f, --filesystem`) допустимы следующие флажки и их комбинации последовательностей:

– `%A` — права доступа в удобной для восприятия человеком форме (`rwx`):

```
$ stat -c%A example.file
-rw-r--r--
```

– `%a` — права доступа в восьмеричной форме (0–7):

```
$ stat -c%a example.file
644
```

– `%B` — размер в байтах каждого блока, отображаемого флажком `%b`:

```
$ stat -c%B example.file
512
```

– `%b` — число занимаемых блоков (см. также флажок `%B`):

```
$ stat -c%b example.file
16
```

– `%D` — номер устройства в шестнадцатеричной форме:

```
$ stat -c%D example.file
30b
```

– `%d` — номер устройства в десятичной форме:

```
$ stat -c%d example.file
779
```

– `%F` — тип файла (например, обычный файл, каталог, ссылка, сокет, блочное устройство, символьное устройство, именованный канал):

```
$ stat -c%F example.file
```

```
regular file (т.е. обычный файл)
```

– %f — права доступа к файлу в «необработанном» (расширенном) шестнадцатеричной виде:

```
$ stat -c%f example.file
81a4 (т.е. 100644 в восьмеричной форме)
```

– %G — действующий (именной) идентификатор группы владельца файла (т. е. название группы):

```
$ stat -c%G example.file
aleksander
```

– %g — числовой идентификатор группы (GID) владельца файла:

```
$ stat -c%g example.file
500
```

– %h — число жёстких ссылок:

```
$ stat -c%h example.file
1
```

– %i — номер индексного дескриптора:

```
$ stat -c%i example.file
786506
```

– %N — имя файла, а для символических ссылок — имя ссылки и оригинального файл с полным путём его месторасположения:

```
$ stat -c%N link.example.file
‘link.example.file’ -> ‘/home/aleksander/example.file’
```

– %n — имя файла:

```
$ stat -c%n example.file
example.file
```

– %o — размер блока для операций ввода и вывода:

```
$ stat -c%o example.file
4096
```

– %s — общий размер файла в байтах:

```
$ stat -c%s example.file
5614
```

– %T — младший номер типа устройства в шестнадцатеричной форме:

```
$ stat -c%T example.file
0
```

– %t — старший номер типа устройства в шестнадцатеричной форме:

```
$ stat -c%t example.file
0
```

– %U — именной идентификатор владельца файла:

```
$ stat -c%U example.file
aleksander
```

– %u — числовой идентификатор владельца файла (UID):

```
$ stat -c%u example.file
500
```

– %X — время последнего доступа к файлу в формате времени UNIX:

```
$ stat -c%X example.file
1114546386
```

– %x — время последнего доступа к файлу в привычном (дата-время) виде:

```
$ stat -c%x example.file
2005-04-26 20:13:06.000000000 +0000
```

– %Y — время последней модификации файла в формате времени UNIX:

```
$ stat -c%Y example.file
1114546659
```

– %y — время последней модификации файла в привычном (дата-время) виде:

```
$ stat -c%y example.file
2005-04-26 20:17:39.000000000 +0000
```

– %Z — время последних изменений файла в формате времени UNIX:

```
$ stat -c%Z example.file
1114546689
```

– %z — время последних изменений файла в привычном (дата-время) виде:

```
$ stat -c%z example.file
2005-04-26 20:18:09.000000000 +0000
```

В качестве формата вывода <формат> для файловых систем (для параметра -f, --filesystem) допустимы следующие флажки и их комбинации последовательностей:

– %a — число блоков доступных не только системному администратору:

```
$ stat -f -c%a example.file
3646377
```

– `%b` — общее число блоков в файловой системе:

```
$ stat -f -c%b example.file
4125127
```

– `%c` — общее число файловых дескрипторов в файловой системе:

```
$ stat -f -c%c example.file
2097152
```

– `%d` — число свободных файловых дескрипторов в файловой системе:

```
$ stat -f -c%d example.file
2086052
```

– `%f` — число свободных блоков в файловой системе:

```
$ stat -f -c%f example.file
3855924
```

– `%i` — идентификатор файловой системы в шестнадцатеричной форме:

```
$ stat -f -c%i example.file
0
```

– `%l` — максимальная длина имён файлов в файловой системе:

```
$ stat -f -c%l example.file
255
```

– `%n` — имя файла :

```
$ stat -f -c%n example.file
example.file
```

– `%s` — оптимальный для файловой системы (операций) размер блока данных:

```
$ stat -f -c%s example.file
4096
```

– `%T` — тип файловой системы в удобном для восприятия человеком виде:

```
$ stat -f -c%T example.file
ext2/ext3
```

– `%t` — тип файловой системы в шестнадцатеричной форме:

```
$ stat -f -c%t example.file
*
```

7.3.19. `file`

Команда `file` предназначена для определения типа файла.

Формат команды:

```
file [<параметры>] <файл...>
```

– `-f <файл>` — считывает из указанного файла список файлов для проверки;

– `-L` — определяет тип файлов, указанных по ссылке;

– `-z` — определяет тип файлов, находящихся в сжатых файлах.

7.3.20. `basename`

Утилита `basename` выделяет из полного имени файла его последнюю часть — локальное имя. По сути отбрасывает весь путь до файла, оставляя имя.

Формат команды:

```
basename <имя файла> [<суффикс>]
```

Не обязательно такое имя файла должно присутствовать в системе. Если указан суффикс, команда удаляет также расширение файла.

Примеры:

```
$ basename /etc/passwd
passwd
$ basename /etc/passwd.bak .bak
passwd
```

7.3.21. `mktemp`

Команда `mktemp` создаёт временный файл.

Формат команды:

```
mktemp [<параметры>] [<шаблон>]
```

Шаблон должен содержать как минимум три символа «X», если шаблон не задан, будет использоваться шаблон `tmp.XXXXXXXXXX`.

Параметры:

– `-d <каталог>` — создавать временный каталог, а не файл;

– `-u`, `--dry-run` — ничего не делать, просто напечатать имя;

– `-q`, `--quiet` — подавлять диагностические сообщения;

– `--suffix=<суффикс>` — добавить `<суффикс>` к `<шаблон>`, он не должен содержать символ «/»;

– `--tmpdir=[<каталог>]` — создавать временный файл в указанном каталоге, если каталог не указан, команда пытается прочитать его из переменной окружения `TMPDIR`, если она недоступна, используется каталог по умолчанию — `/tmp`.

7.4. Утилиты работы с файловыми системами

В данном разделе перечислены особенности работы с устройствами — носителями данных и файловыми системами.

7.4.1. `/etc/fstab`

Конфигурационный файл, который содержит информацию о различных файловых системах и устройствах хранения информации компьютера; описывает, как диск (раздел) будет использоваться или как будет интегрирован в систему.

7.4.1.1. Структура

Каждая запись имеет следующие поля (которые разделяются пробелом или табуляцией):

`<ФС> <каталог> <тип> <параметры> <дамп> <приоритет>`

– `<ФС>` — файловая система, указывается имя монтируемого устройства (файл устройства) или его UUID;

– `<каталог>` — каталог, в который нужно смонтировать устройство;

– `<тип>` — тип файловой системы, например, ext4. Ключ `auto` позволяет включить автоматическое определение файловой системы;

– `<параметры>` — параметры, которые указываются в ключе `-o` команды `mount`, список основных параметров приведён в таблице 8;

– `<дамп>` — выполнять команду `dump` для данной файловой системы (резервная копия);

– `<приоритет>` — число, которое определяет, в каком порядке утилита `fsck` должна производить проверку файловых систем. Возможные значения 0, 1 и 2. Файловые системы со значением, равным 0, не будут проверены утилитой `fsck`. У корневой системы должен быть наибольший приоритет, 1, остальные файловые системы должны иметь приоритет 2.

Таблица 8 – Список основных параметров монтирования

Параметр	Описание
`async`	Все операции ввода и вывода файловой системы будут выполняться асинхронно

Параметр	Описание
atime	Включает запись информации о последнем времени доступа при каждом чтении файла
auto	Файловая система монтируется при загрузке автоматически или после выполнения команды `mount -a`
defaults	Использовать значения по умолчанию. Соответствует набору `rw, suid, dev, exec, auto, nouser, async`
dev	Различает файловые системы символьных и блочных устройств. Должен обязательно использоваться для корневого каталога и `chroot`
dirsync	Все каталоги обновляются в пределах файловой системы с целью синхронизации информации. Это происходит при следующих системных вызовах: `creat`, `link`, `unlink`, `symlink`, `mkdir`, `rmdir`, `mknod` и `rename`;
exec	Позволяет исполнять бинарные файлы на разделе диска. Установлено по умолчанию
group	Позволяет обычному пользователю (т. е., не системному администратору) монтировать файловую систему если он является членом одной из групп, которой принадлежит устройство
mand	Позволяет принудительно заблокировать данную файловую систему
_netdev	Файловая система, находящаяся на устройстве, которому требуется сетевой доступ (используется для предотвращения попыток системы монтировать эту файловую систему пока доступна сеть)
noatime	Отключает запись информации о последнем времени доступа при каждом чтении файла
noauto	Файловая система может быть смонтирована только вручную
nodev	Данный параметр предполагает, что на монтируемой файловой системе не будут созданы файлы устройств (`/dev`)
noexec	Запрещает исполнение любых двоичных файлов смонтированной файловой системы
nomand	Не позволяет принудительно блокировать данную файловую систему
nosuid	Запрещает операции с SUID и SGID битами

Параметр	Описание
`nouser`	Запрещает обычному пользователю монтировать файловую систему, что принято по умолчанию
`owner`	Позволяет обычному пользователю монтировать файловую систему, если он является владельцем устройства
`relatime`	Включает запись информации о последнем времени доступа при чтении файла, если предыдущее время доступа меньше времени изменения файла
`remount`	Пробует перемонтировать уже смонтированную файловую систему. Это может быть необходимо, когда требуется поменять флаги режима доступа. Например, файловая система с режимом «только для чтения» может быть преобразована и после перемонтирования допускает режим «чтение/запись». Это действие не изменяет устройство или точку монтирования
`ro`	Монтирует файловую систему только для чтения
`rw`	Монтирует файловую систему для чтения/записи
`suid`	Разрешить операции с SUID и SGID битами. В основном используются, чтобы позволить пользователям выполнять бинарные файлы со временно приобретёнными привилегиями для выполнения определённой задачи
`sync`	Все операции ввода/вывода должны выполняться синхронно
`users`	Разрешает любому пользователю монтировать файловую систему
`user`	Разрешает указанному пользователю монтировать файловую систему

7.4.2. `mount`

Команда `mount` монтирует файловую систему.

Формат команды:

```
mount [-lhV]
```

или

```
mount -a [-fFnrsvw] [-t <тип ФС>] [-O <список параметров>]
```

или

```
mount [-fnrsvw] [-o <параметры> [,...]]
<устройство>|<каталог>
```

или

```
mount [-fnrsvw] [-t <тип ФС>] [-o параметры] <устройство>
<каталог>
```

Все файлы составляют иерархическую файловую структуру, которая подобна растущему дереву имеет ветки (каталоги) и листья (файлы в каталогах). Корень этого большого дерева обозначается как «/». Физически файлы могут располагаться на различных устройствах. Команда `mount` служит для подключения файловых систем разных устройств к этому большому дереву. Также существует противоположная команда `umount`, которая выполняет размонтирование (отключение) файловых систем.

Наиболее часто встречающаяся форма команды `mount` выглядит следующим образом:

```
mount -t <тип ФС> <устройство> <каталог>
```

Такая команда предлагает ядру смонтировать (подключить) файловую систему указанного типа `<тип ФС>`, расположенную на устройстве `<устройство>`, к заданному каталогу `<каталог>`, который часто называют точкой монтирования. Предыдущее содержимое, владелец и режим доступа к каталогу `<каталог>` становятся недоступными (исчезают), а вновь появившиеся продолжают действовать, пока файловая система `<устройство>` смонтирована (подключена) к `<каталог>`.

Следующие три формы вызова этой команды не связаны с каким-либо монтированием:

```
mount -h
```

выводит на экран помощь.

```
mount -V
```

выводит информацию о версии программы.

```
mount [-l] [-t <тип ФС>]
```

выводит список всех смонтированных файловых системы с определённым типом `<тип ФС>`. В этом списке параметр `-l` добавляет некоторые описания, например, ext2, ext3, XFS. Если тип файловой системы не указан, выводит все смонтированные файловые системы.

Возможно осуществлять монтирование одного каталога в другой при помощи команды:

```
mount --bind <старый каталог> <новый каталог>
```

После выполнения этой команды одно и то же содержимое становится доступным из двух точек. Таким же образом можно перемонтировать отдельный файл.

Данная форма команды монтирует только часть некоторой файловой системы, без подмонтированных внутри ранее (или позднее) файловых систем. Полную файловую иерархию, включая подмонтированные внутри файловые системы, можно смонтировать при помощи следующего вызова:

```
mount --rbind <старый каталог> <новый каталог>
```

Следует помнить, что новая точка монтирования файловой системы, например, `<новый каталог>`, наследует свойства (например, права владельцев), от первоначальной точки монтирования `<старый каталог>`, которые не могут быть изменены через параметр `-o`, пока действуют `--bind`/`--rbind`.

Существует возможность атомарно перемещать (т. е. операция или выполняется до конца, или вообще не выполняется) смонтированное дерево каталогов в иное место. Для этого используют следующий вызов:

```
mount --move <старый каталог> <новый каталог>
```

Такая файловая система, как `proc`, не ассоциирована ни с каким специальным устройством, поэтому когда её монтируют, условное ключевое слово `proc` может быть использовано взамен спецификации устройства.

Большинство устройств определяется именем файла (для специальных блочных устройств), например, `/dev/sda1`, однако существуют и другие описания. Например, в случае монтирования некоего NFS устройства `device` можно встретить такое его описание: `knuth.cwi.nl:/dir`. Также есть возможность указать специальное блочное устройство, используя его метку тома или UUID.

Ключевую роль в процессе монтирования играет файл `/etc/fstab`, строки которого обычно характеризуют подключаемые устройства с соответствующими для них параметрами. Этот файл используется в следующих случаях:

1) пытается смонтировать все файловые системы типа `<тип ФС>`, которые перечислены в файле `/etc/fstab` (с соответствующим типом `<тип ФС>` и/или имеющие или не имеющие соответствующие параметры), за исключением тех, чьи строки описаний содержат ключевое слово `noauto`. С дополнительным параметром

-F операция монтирования будет выполняться параллельно и файловые системы монтироваться одновременно (обычно встречается в сценариях загрузки):

```
mount -a [-t <тип ФС>] [-O <список параметров>]
```

2) если монтируемая файловая система упоминается в файле /etc/fstab, то для команды mount достаточно в параметрах указывать только устройство или точку монтирования;

3) обычно только системный администратор может монтировать файловые системы, но, если в строке описания файла /etc/fstab содержится параметр user или users, соответствующую файловую систему может монтировать любой пользователь.

Так, задавая в строке файла fstab следующее описание

```
/dev/cdrom /cd iso9660 ro,user,noauto,unhide
```

любой пользователь сможет монтировать файловую систему iso9660 для устройства CD-ROM с помощью команды

```
mount /dev/cdrom
```

или

```
mount /cd
```

Лишь тот пользователь, который смонтировал файловую систему, может её размонтировать. Если необходимо, чтобы любой пользователь мог выполнить операцию размонтирования, тогда в строках описания файла /etc/fstab используйте параметр users вместо user.

Параметр owner подобен параметру user, с тем ограничением, что пользователь должен быть владельцем специального файла (устройства). Этот параметр может быть полезна, например, для /dev/fd если сценарий начальной регистрации будет назначать пользователя владельцем этого устройства. Параметр group подобен параметру user, с тем лишь ограничением, что пользователь должен быть членом группы, которой принадлежит специальный файл (устройство).

Программы mount и umount поддерживают список текущих смонтированных файловых систем в файле /etc/mtab. Запущенная без аргументов, mount выводит на экран этот список.

Когда файловая система proc монтируется, к примеру, на точку /proc, содержание файлов /etc/mtab и /proc/mounts весьма схожее между собой. При этом в первом из них содержится несколько больше информации, так как здесь дополнительно указываются параметры монтирования, хотя это не всегда соответствует

действительности (если корневая файловая система смонтирована в режиме «только чтение», например, во время ремонта файловой системы, то файл /etc/mtab не может быть создан и его использование приведет к получению недостоверной информации). Допустима возможная замена /etc/mtab символьной ссылкой на /proc/mounts особенно когда очень много смонтированных ресурсов — это должно существенно улучшить работу через такую символьную ссылку. При этом будет утеряна некоторая информация, специфическая работа с loop устройствами станет менее удобной и, кроме того, невозможным станет использование параметра user.

Полный набор параметров, который будет использован при работе команды mount в первую очередь включает параметры, извлекаемые из таблицы файла /etc/fstab для определенной файловой системы, затем добавляются параметры, которые задаются аргументом -o и, наконец, если они указаны, добавляются параметры -r или -w.

Параметры, которые можно использовать с командой mount:

– -v — подробно информирует о выполняемых действиях;

– -a — монтирует все файловые системы из /etc/fstab;

– -F — подключать файловые системы на различных устройствах или на серверах NFS параллельно, что благоприятно сказывается на производительности;

– -f — пытается делать все возможное для выполнения системного вызова, полезно для диагностики;

– -i — не вызывать помощника монтирования /sbin/mount.<filesystem> даже если он существует;

– -r — монтирует файловую систему в режиме «только для чтения»;

– -w — монтирует файловую систему в режиме «чтения/запись»;

– -U uuid — монтирует раздел по его UUID;

– -t <тип ФС> — аргумент следующий за -t указывает тип файловой системы;

– -o — указывает список параметров, разделенных запятыми. Некоторые из этих параметров полезны, когда они представлены в файле /etc/fstab. Список основных параметров перечислен в таблице 8, для каждой конкретной файловой системы существуют свои параметры, которые перечислены ниже;

– --bind — выполняет дополнительное монтирование поддерева каталогов еще в одну точку (после этого содержимое становится доступным из обоих точек);

– --move — перемещает поддерево в некоторое иное место.

Для каждой файловой системы существуют свои собственные параметры монтирования. Ниже рассмотрены основные параметры файловых систем iso9660 (файловая система для компакт-дисков), ext3 и ext4.

7.4.2.1. Параметры монтирования для ext2

Файловая система ext2 является устаревшей, однако она служила прототипом для ext3 и ext4.

– `acl / noacl` — поддерживать списки управления доступом или нет;

– `bsddf / minixdf` — устанавливает поведение системного вызова `statfs` в случае предоставления статистики по файловой системе. При этом `minixdf` возвращает в поле `f_blocks` общее количество блоков, используемых файловой системой, а `bsddf` (параметр по умолчанию) — отбрасывает часть блоков, которые необходимы файловой системе ext2, но не применяются для хранения файлов, например:

```
# mount /k -o minixdf; df /k; umount /k
Filesystem 1024-blocks Used Available Capacity Mounted on
/dev/sda6 2630655 86954 2412169 3% /k
# mount /k -o bsddf; df /k; umount /k
Filesystem 1024-blocks Used Available Capacity Mounted on
/dev/sda6 2543714 13 2412169 0% /k
```

Следует отметить, что в этом примере к параметрам, заданным в командной строке, добавляются соответствующие параметры файла `/etc/fstab`;

– `check` — проверяет файловую систему (блоки и индексные дескрипторы) во время монтирования;

– `check=none / nocheck` — не выполняет никаких проверок во время монтирования;

– `debug` — выводит отладочную информацию при каждом (пере)монтировании;

– `errors=continue / errors=remount-ro / errors=panic` — определяет поведение в случае обнаружения ошибок:

1) `errors=continue` — игнорирует ошибки, однако помечает файловую систему как некорректную, после этого монтирование продолжается;

2) `errors=remount-ro` — перемонтирует файловую систему в режим «только для чтения»;

3) errors=panic — аварийно завершает процесс монтирования и блокирует работу системы;

установка по умолчанию задана в суперблоке файловой системы и может быть изменена с помощью tune2fs;

– grpid или bsdgroups / nogrpid или sysvgroups — эти параметры определяют, какой идентификатор группы получит вновь созданный файл. Если указан параметр grpid, файл принимает GID каталога, в котором он создан; иначе (по умолчанию или второй вариант) файл принимает GID текущего процесса, если только для каталога не установлен SGID, поскольку в этом случае берется GID родительского каталога и может также приобрести SGID;

– grpquota / noquota / quota / usrquota — эти параметры доступны, но будут проигнорированы;

– nobh — не присоединять buffer_heads к файлу кешируемых страниц (pagecache);

– nouid32 — не использовать 32-битные UID-ы и GID-ы. Эта возможность введена для поддержки очень старых ядер, которые хранят и ожидают только 16-битные значения;

– oldalloc или orlov — использовать старый алгоритм распределения или алгоритм Орлова для новых индексных дескрипторов. По умолчанию используется алгоритм Орлова;

– resgid=n и resuid=n — файловая система ext2 резервирует определенный процент доступного дискового пространства под свои нужды (по умолчанию 5%, см. mke2fs и tune2fs). Эти параметры определяют кто может использовать зарезервированные блоки. Так, для resgid зарезервированные блоки может использовать любой, если он принадлежит к группе n. В случае resuid эти блоки может использовать любой, чей UID равен n;

– sb=n — взамен блока 1, в качестве суперблока используется n блок. Это может быть полезно, когда файловая система повреждена. Для номера блока здесь используется 1k разрядов. Так, если вы пожелаете использовать логический блок 32768 на файловой системе с 4k блоками, используйте sb=131072;

– user_xattr / nouser_xattr — поддерживать дополнительные пользовательские атрибуты или нет.

7.4.2.2. Параметры монтирования для ext3

Файловая система ext3 поддерживает функцию журналирования в отличие от ext2. Доступные все параметры фаловой системы ext2, а также следующие:

– `journal=update` — обновляет журнал файловой системы ext3 к текущему формату;

– `journal=inum` — если журнал уже существует, этот параметр игнорируется. Иначе, она определяет номер индексного дескриптора (inode), который будет исполнять роль журнала файловой системы ext3; для ext3 будет создан новый журнал, который заменит прежнее содержимое для файла, чей номер inode равен `inum`;

– `noload` — не загружать журнал файловой системы ext3 при монтировании;

– `data=journal|ordered|writeback` — определяют режим журналирования для файловых данных. Метаданные журналируются всегда. Чтобы на корневой файловой системе использовать другие режимы, кроме `ordered`, передайте необходимый режим ядру в виде параметра, например `rootflags=data=journal`. Поддерживаются следующие режимы журналирования:

1) `journal` — до начала изменений (сохранения) в основной файловой системе, все данные помещаются сначала в журнал;

2) `ordered` — режим по умолчанию. Все изменения сначала происходят в основной файловой системе, а затем их метаданные регистрируются в журнале;

3) `writeback` — оперирование данными без резервирования — данные сохраняются в основной файловой системе, после этого метаданные регистрируются в журнале. Этот параметр считается высокопроизводительным. Он гарантирует внутреннюю целостность файловой системы, однако это может приводить к перезаписи старыми данными информации в файлах после аварийного завершения работы системы и последующего процесса восстановления из журнала;

– `commit=nrsec` — синхронизировать все данные и метаданные каждые `nrsec` секунд. По умолчанию это значение равно 5 секундам. Ноль означает взять значение по умолчанию.

7.4.2.3. Параметры монтирования для ext4

Для ext4 доступны следующие параметры из файловых систем ext3 и ext2: `journal_dev`, `noload`, `data`, `commit`, `orlov`, `oldalloc`, `[no]user_xattr` `[no]acl`, `bsddf`, `minixdf`, `debug`, `errors`, `data_err`, `grpid`, `bsdgroups`, `nogrpid sysvgroups`, `resgid`, `resuid`, `sb`, `quota`, `noquota`, `grpquota`, `usrquota`, `[no]bh`.

Дополнительные параметры для ext4:

– `journal_checksum` — включить контрольные суммы для транзакций журнала. Это позволяет при помощи e2fsck или ядра выявлять нарушения в работе;

– `journal_async_commit` — блок транзакции может быть записан на диск без ожидания блоков дескриптора;

– `journal=update` — обновить журнал файловой системы под текущий формат;

– `barrier=0 / barrier=1 / barrier / nobarrier` — включить/выключить барьеры в коде `jbd`. 0 — выключить, 1 — включить. Барьеры принуждают выполнять записи в журнал по порядку. Барьеры включены по умолчанию;

– `inode_readahead=n` — контролирует максимальное количество индексных дескрипторов, которые будут считываться алгоритмом упреждающего чтения в кэш. Стандартное значение — 32 блока;

– `stripe=n` — количество блоков, которые `mballoc` будет пытаться использовать для выделения места и выравнивания. Для RAID5/6 это должно быть равно `<количество дисков с данными> * <размер раздела RAID>` в блоках файловой системы;

– `delalloc` — откладывать выделение блоков до завершения операций записи;

– `nodelalloc` — отключить отложенное выделение блоков;

– `max_batch_time=usec` — максимальное время ожидания для объединения операций в одну группу. По умолчанию это 15 мс (15000 мкс);

– `min_batch_time=usec` — максимальное время ожидания для объединения операций в одну группу. Должно быть меньше `max_batch_time`;

– `journal_ioprio=prio` — приоритет ввода-вывода журнала от 0 до 7, 0 — наивысший приоритет. По умолчанию 3, что немного выше приоритета обычных операций ввода-вывода;

– discard/nodiscard — контролирует, нужно ли отправлять команды discard/TRIM блочному устройству, на котором располагается файловая система, когда блоки данных освобождаются. Может быть полезно для SSD;

– nouid32 — отключает 32-разрядные UID и GID;

– resize — разрешить изменять размер файловой системы;

– block_validity/noblock_validity — выключить или выключить возможность отслеживания состояния метаданных ядром. Данный параметр предназначен для отладки и по умолчанию отключён;

– dioread_lock/dioread_nolock — использовать или не использовать блокировку чтения DIO. Если блокировка выключена (указан параметр dioread_nolock), перед записью буфера будет создаваться неинициализированный экстент, который будет проинициализирован после завершения операций ввода-вывода. Это позволяет избежать использования мьютекса для индексного дескриптора, что улучшает масштабируемость на высокоскоростных устройствах хранения. Однако, это не работает совместно с параметром nobh и монтирование завершится с ошибкой;

– i_version — включить поддержку 64-разрядных индексных дескрипторов. По умолчанию выключено.

7.4.2.4. Параметры монтирования для tmpfs

Данный тип файловых систем предназначен для временных каталогов. Размещается в оперативной памяти.

Следующим параметрам в качестве приставок доступны k, m или g, которые можно изменить при перемонтировании.

– size=<байт> — переопределяет максимальный размер файловой системы заданный по умолчанию. Размер задается в байтах и округляется вниз к целому числу страниц. По умолчанию используется половина памяти;

– nr_blocks=<байт> — устанавливает количество блоков;

– nr_inodes=<байт> — устанавливает количество индексных дескрипторов;

– mode=<байт> — устанавливает начальные уровни доступа к корневому каталогу.

7.4.2.5. Коды завершения

`mount` возвращает определённые коды по окончании свой работы, которые могут составлять общий результирующий код. С помощью побитовой логической операции «И» следующие отдельные коды можно выделить из возвращаемого общего кода.

– 0 — успешное выполнение;

– 1 — некорректные вызов или права доступа;

– 2 — системная ошибка (нехватка памяти, недостаточно ресурсов для ветвления процесса, нет доступных петлевых устройств);

– 4 — внутренняя ошибка `mount` или отсутствует поддержка nfs в `mount`;

– 8 — работа прервана пользователем;

– 16 — проблема записи или блокирования `/etc/mtab`;

– 32 — ошибка монтирования;

– 64 — некоторые из операций монтирования — успешны.

7.4.2.6. Файлы

– `/etc/fstab` — таблица файловых систем;

– `/etc/mtab` — таблица смонтированных файловых систем;

– `/etc/mtab~` — файл блокировки;

– `/etc/mtab.tmp` — временный файл;

– `/etc/filesystems` — список используемых типов файловых систем.

7.4.2.7. Примеры

Монтировать устройство `/dev/cdrom` в каталог `/mnt/cdrom`

```
# mount /dev/cdrom /mnt/cdrom
```

Устройство `/dev/cdrom` монтируется в каталог `/mnt/cdrom`, если он существует. Начиная от момента монтирования и пока пользователь не отмонтирует файловую систему (или туда не будет смонтировано что-то иное) в каталоге `/mnt/cdrom` будет содержаться дерево каталогов устройства `/dev/cdrom`; те файлы, и подкаталоги, которые раньше находились в `/mnt/cdrom`, сохранятся, но будут недоступны до размонтирования устройства `/dev/cdrom`.

Монтирование EXT4 дисков/разделов:

```
# mount -o noatime,users,rw /dev/sda1 /mnt
```

Если имеется образ диска в виде ISO-файла (образ CD/DVD), то для его монтирования нужно указать параметр `-o loop`. В данном примере `file.iso` — имя файла образа, а `/mnt/iso` — точка монтирования:

```
# mount -o loop -t iso9660 file.iso /mnt/iso
```

Монтирование сетевых дисков NFS:

```
# mount 172.22.2.1:/mnt/iso/ /mnt/iso/
```

Пример монтирования сетевых SMB-ресурсов:

```
# mount -t cifs -o username=user,password=pass //server/dir
/mnt/localdir/
```

Параметр `-t cifs` можно не указывать, если имя монтируемого устройства вида `//server/dir`.

Команда `mount` с ключом `--bind` или с коротким ключом `-B` применяется для создания синонима каталога в дереве файловой системы. К примеру, команда:

```
# mount --bind /mnt/cdrom/Files /var/ftp/cdrom
```

позволит обращаться к файлам из `/mnt/cdrom/Files` через путь `/var/ftp/cdrom`, где `/var/ftp/cdrom` — некий уже существующий (возможно, пустой) каталог (его настоящее содержимое будет недоступно до момента размонтирования). Можно также вместо отдельного параметра `--bind` написать `-o bind`, что будет иметь аналогичное действие. Также это позволяет добавить правило в файл `/etc/fstab` для монтирования при старте системы:

```
/olddir /newdir none bind
```

7.4.3. `umount`

Команда `umount` применяется для размонтирования файловых систем, предварительно смонтированных посредством вызова утилиты `mount`. Для её использования требуются привилегии системного администратора. При выключении компьютера вызывается автоматически.

Формат команды:

```
umount <точка монтирования>|<файл устройства>
```

Пример:

```
umount /dev/cdrom
```

7.4.4. `mkfs`

Команда `mkfs` используется для создания файловых систем.

Формат команды:

```
mkfs [<параметры>] [<параметры ФС>] <устройство>
```

`mkfs` используется для создания файловой системы на некотором устройстве, обычно в разделе жёсткого диска. В качестве аргумента `<устройство>` для файловой системы может выступать или название устройства (например, `/dev/hda1`, `/dev/sdb2`) или точка монтирования (например, `/`, `/usr`, `/home`).

По окончании работы `mkfs` возвращает 0 — в случае успеха, а 1 — при неудачной операции.

В общем случае, `mkfs` является простым конечным интерфейсом к доступным под модулям создания файловых систем, в которых вторая часть сложных имён (`mkfs.fstype`) как раз и определяет вызываемый модуль.

Параметры:

– `-V` — подробно информирует происходящее, включая все выполняемые специфические команды файловой системы. Если указать этот параметр более одного раза, то это запретит реальное выполнение любых специфических команд файловой системы. Использовать этот параметр целесообразно во время тестирования;

– `-t <тип ФС>` — указывает тип создаваемой файловой системы. Если этот параметр не указан, тогда, по умолчанию, принимается тип файловой системы ext2;

– `<параметры ФС>` — передаёт модулю создания специфической файловой системы параметры в виде списка. Следует отметить, что нет гарантии в том, что следующие перечисленные параметры будут поддерживаться большинством модулей создания файловых систем;

– `-c` — перед созданием файловой системы проверяет наличие сбойных блоков на устройстве;

– `-l <файл>` — считывает список сбойных блоков из указанного файла `<файл>`. Для составления подобного списка, можно выполнить предварительную проверку, например, с помощью программы `badblocks`;

– `-v` — подробно комментирует происходящее.

7.4.5. `blkid`

Команда `blkid` предназначена для работы с блочными устройствами и позволяет получить UUID дискового раздела:

```
# blkid /dev/sda3
```

```
/dev/sda3: UUID="8f4b9853-93f6-44f0-a3e1-24eff52fc409"
TYPE="ext4"
```

Она необходима, когда нужно знать UUID раздела для его использования в `/etc/fstab`.

Для того чтобы получить только UUID, используйте команду:

```
# blkid /dev/sda3 | cut -d\" -f2
8f4b9853-93f6-44f0-a3e1-24eff52fc409
```

Если команда вызвана без параметров, отображает информацию по всем устройствам.

7.4.6. `lsblk`

Утилита `lsblk` отображает список доступных блочных устройств.

Формат команды:

```
lsblk [-l]
```

По умолчанию список выводится в виде дерева, чтобы получить простой список, нужно использовать параметр `-l`.

7.4.7. `fdisk`

Разбиение диска производится с помощью программы `fdisk`. Программа `fdisk` имеет интерфейс командной строки.

Для просмотра разделов диска, используйте команду:

```
# fdisk -cul /dev/sda
```

Для изменения разметки запустите команду с параметрами `-cu`:

```
# fdisk -cu /dev/sda
```

Команда работает в интерактивном режиме, для получения справки по командам нажмите `m`.

Параметр `-c` выключает режим совместимости с DOS, а `-u` — использует разметку диска в секторах.

Пример создания нового основного раздела:

1) запустите программу `fdisk` и создайте новый раздел:

```
# fdisk -cu /dev/sda
//создание нового раздела
Команда (m для справки): n
//Создаётся новый раздел размером 1 Гб
```

```
    Первый сектор (1048580096-1953523711, по умолчанию
1048580096): [Enter]
    Используется значение по умолчанию 1048580096
    Last сектор, +секторы or +size{K,M,G}
(1048580096-1048582143, по умолчанию 1048582143): +1G

    //записать изменения
    Команда (m для справки): w
```

2) после записи изменений нужно перезагрузить компьютер:

```
# reboot
```

3) после того, как новый раздел создан, нужно создать для него файловую систему:

```
# mkfs.ext4 /dev/sda3
```

П р и м е ч а н и е. Аккуратно используйте названия дисковых разделов, так как они нумеруются, можно по ошибке создать новую файловую систему не на том диске, тем самым удалив важные данные. Для того чтобы убедиться, что данный диск — именно тот, который нужен, используйте команду `df` для просмотра дисковых разделов, их точек монтирования и размеров;

1) создайте новый каталог — точку монтирования для нового раздела:

```
# mkdir /mydevice
```

2) используя команду `blkid`, получите UUID данного раздела:

```
# blkid /dev/sda3
/dev/sda3: UUID=”8f4b9853-93f6-44f0-a3e1-24eff52fc409”
TYPE=”ext4”
```

3) запись о новой файловой системе нужно добавить в `/etc/fstab`, чтобы она монтировалась при запуске системы:

```
UUID=<uuid> /mydevice ext4 defaults 1 2
```

Здесь, `<uuid>` — UUID, полученный на предыдущем шаге.

4) запустите процесс монтирования всех дисков:

```
# mount -a
```

Или монтирование только данного раздела:

```
# mount /mydevice
```

7.4.8. badblocks

Утилита badblocks сканирует файловые системы на наличие плохих секторов.

Формат команды:

badblocks [<параметры>] <устройство> [<последний блок>] [<первый блок>]

Параметры:

– -b <размер блока> — задаёт используемый на диске размер блоков в байтах.

– -c <порция блоков> — указывает порцию блоков, которые будут проверены за раз;

– -f — выполнять проверку даже в том случае, если это может повредить систему (не рекомендуется);

– -i <входной файл> — читает список уже существующих (найденных ранее) сбойных блоков;

– -o <исходящий файл> — сохраняет список сбойных блоков в указанный исходящий файл;

– -p <число раз поиска> — повторяет поиск сбойных блоков на одном и том же пространстве диска указанное в параметре число раз;

– -t <проверяемый образец> — задаёт проверяемый образец для операций чтения/записи блоков диска. <проверяемый образец> может быть или числовым значением лежащим между 0 и ULONG_MAX-1 включительно, или словом random, которое указывает на то, что блок должен быть заполнен образцом в виде случайных бит. Для режима чтения/записи -w или безопасного -n режима проверки может быть указан один или более проверяемых образцов, каждый из которых должен быть описан параметром -t. Для режима только-чтение может быть задан исключительно один образец и это не может быть образец random. Проверка с образцом в режиме только-чтение означает, что указанный образец должен быть предварительно записан на диск — если это невозможно сделать, тогда при тестировании будет пропущено большое количество блоков. Если указана проверка по множеству образцов, тогда все блоки будут сначала проверены по одному образцу, а затем по следующим;

– -n — заставляет использовать режим чтения-записи, который не разрушает данные. Этот режим используется по умолчанию для выполнения проверки диска.

Этот параметр не должен сочетаться с параметром `-w`, поскольку их действия исключают друг друга;

– `-s` — отображает ход процесса проверки, выводя номера блоков в том порядке, как они проверялись;

– `-v` — подробно сообщать о происходящем;

– `-w` — для проверки используется режим реальной записи. С этим параметром команда `badblocks` выполняет поиск сбойных блоков, сначала записывая специальные образцы (0xaa, 0x55, 0xff, 0x00) в каждый проверяемый блок устройства, затем читая каждый блок и сравнивая его содержимое делает вывод. Этот параметр не может сочетаться с параметром `-n`, так как их действия исключают друг друга.

7.4.9. `mke2fs`

Утилита `mke2fs` — позволяет создать раздел типа ext2/ext3/ext4 на размеченной области жёсткого диска. Использует значения по умолчанию, перечисленные в файле `/etc/mke2fs.conf`.

Формат команды:

```
mke2fs [<параметры>] <устройство>
```

Параметры:

– `-t <тип ФС>` — тип создаваемой файловой системы;

– `-b` — размер одного блока данных в байтах. Обычные значения: `1024`, `2048` или `4096` байт. Если параметр опущен, то используется значение, основанное на собранных о жёстком диске данных. Если значение неверно, то используется наиболее подходящее к этому числу значение;

– `-c` — проверка на наличие повреждённых блоков перед созданием раздела. Если параметр указан дважды, то будет проведено тщательное сканирование блоков (проверка на чтение и запись против обычной проверки чтением);

– `-F` — принудительное создание файловой системы, даже если указанное устройство вовсе не является разделом. При указании данного параметра дважды, утилита игнорирует сообщение о том, что данное устройство используется.

7.4.10. `tune2fs`

Утилита `tune2fs` используется для изменения параметров файловой системы.

Формат команды:

```
tune2fs [<параметры>] <устройство>
```

Параметры:

– `-e <действие при ошибке>` — определяет действие, которое необходимо выполнить при возникновении ошибки. В любом случае необходимо запустить проверку файловой системы при помощи утилиты `fsck` при последующей перезагрузке. Возможные действия:

1) `continue` — продолжить работу (по умолчанию);

2) `remount-ro` — перемонтировать файловую систему в режиме только для чтения;

3) `panic` — вызвать панику ядра (остановить систему);

– `-f` — принудительно завершить операцию даже в случае возникновения ошибок. Это полезно при удалении возможности журналирования для файловой системы, у которой есть внешний журнал, но он на данный момент недоступен;

– `-g <группа>` — задать группу, которая может использовать зарезервированные блоки файловой системы. Параметр <группа> — это идентификатор группы пользователей или её имя;

– `-i <интервал проверок>[d|m|w]` — задать максимальное время для повторной проверки файловой системы. Окончание `d` (по умолчанию) означает, что интервал задан в днях, `m` — месяцах, `w` — неделях;

– `-j` — добавить журнал для файловой системы (только ext3 или ext4);

– `-J <параметры журнала>` — задать параметры журнала. Обычно по умолчанию заданы оптимальные параметры. Возможные параметры:

1) `size=<размер>` — размер журнала в мегабайтах;

2) `device=<внешний журнал>` — назначить другое блочное устройство для хранения журнала (внешний журнал);

– `-l` — показать содержимое суперблока;

– `-L <метка тома>` — задаёт метку тома;

– `-m <процент зарезервированного пространства>` — процент зарезервированных блоков под нужны файловой системы;

– `-M <последний каталог монтирования>` — задать последний каталог монтирования для файловой системы;

– `-o [^]<параметр монтирования...>` — задать или снять параметр монтирования файловой системы. Стандартные параметры монтирования могут быть перекрыты содержимым файла `/etc/fstab` или аргументами командной строки команды `mount`. Можно перечислить несколько параметров через запятую. Если

задана приставка «^», параметр удаляется. Подробное описание параметров монтирования приведено в разделе 7.4.2;

– `-O [^]<возможность...>` — задать или снять возможность файловой системы. Если задана приставка «^», параметр удаляется. Файловая система поддерживает следующие возможности:

1) `dir_index` — использовать хэшированные b-деревья для ускорения просмотра больших каталогов;

2) `filetype` — сохранять информацию о типе файла в записях каталогов;

3) `has_journal` — использовать журналирование для обеспечения целостности файловой системы (аналогично параметру `-j`);

4) `sparse_super` — ограничить количество резервных суперблоков для сохранения пространства на больших устройствах;

– `-r <количество зарезервированных блоков>` — установить количество зарезервированных блоков файловой системы;

– `-s [0|1]` — включить возможность `sparse_super` для экономии пространства (аналогично `-O sparse_super`);

– `-T <время последней проверки>` — установить время последней проверки. Формат: `ГГГГММДД[ЧЧММ[СС]]`;

– `-u <пользователь>` — установить пользователя, который может использовать зарезервированные блоки файловой системы. Пользователя можно задать как по уникальному идентификатору, так и по имени;

– `-U <UUID>` — задать универсальный уникальный идентификатор файловой системы. Формат уникального идентификатора — набор шестнадцатеричных символов, разделённых дефисами, например: «e1b9d5a2-f162–91ae-9ece-9820afc76f46». Также можно использовать следующие ключевые слова:

1) `clear` — удалить UUID файловой системы;

2) `random` — назначить новый UUID случайным образом;

3) `time` —назначить новый UUID, основываясь на текущем времени.

UUID может впоследствии использоваться командами `mount`, `fsck` и в файле `/etc/fstab` вместо указания пути до блочного устройство. Рекомендуется всегда использовать UUID, когда это возможно.

7.4.11. fsck

Команда fsck проверяет и устраняет ошибки в файловой системе.

Формат команды:

```
fsck [<параметры>] [<файловая система>]
```

fsck используется для проверки и, в случае необходимости, исправления ошибок одной или нескольких файловых систем. <файловая система> может быть именем устройства (например, /dev/sda7), точкой монтирования этого блочного устройства (/, /mnt/sda7 и т. д.), меткой раздела или UUID-индексом. Обычно fsck параллельно проверяет данные на разных физических дисках, чтобы сократить общее время, необходимое для полной проверки всех дисков.

Если файловых систем не указано и не указан параметр -A, то fsck по порядку проверит файловые системы, указанные в /etc/fstab. Это эквивалентно параметру -As. Код, возвращаемый fsck, является суммой следующих условий:

– 0 — нет ошибок;

– 1 — ошибки файловой системы исправлены;

– 2 — необходима перезагрузка системы;

– 4 — ошибки файловой системы не исправлены;

– 8 — в процессе проверки произошли ошибки;

– 16 — неверное использование команды либо синтаксическая ошибка;

– 32 — fsck была прервана пользователем;

– 128 — ошибка разделяемых объектов.

Код, возвращаемый fsck при проверке нескольких файловых систем, получается с помощью применения побитовой операции ИЛИ к кодам, возвращаемым каждой проверкой.

Фактически fsck представляет собой оболочку для различных средств проверки файловой системы. Она определяет тип используемой на носителе файловой системы и запускает необходимые программы для проверки.

7.4.12. df

Команда df выводит сведения о свободном месте на дисковых разделах системы. При запуске без аргументов df выдаёт отчёт по доступному и использованному пространству для всех смонтированных файловых систем (всех типов). В против-

ном случае, df для каждого файла, указанного в командной строке, выдаёт отчёт по файловой системе, которая его содержит.

Формат команды:

df [<параметры>] [<файл...>]

При запуске без аргументов df выдаёт отчёт по доступному и использованному пространству для всех смонтированных файловых систем (всех типов). В противном случае, df для каждого файла, указанного в командной строке, выдаёт отчёт по файловой системе, которая его содержит.

Параметры:

– -a, --all — включает в список файловых систем те, которые имеют размер в 0 блоков, и которые по умолчанию опускаются;

– --block-size=<размер> — выдаёт размеры в блоках заданного размера (байт);

– -h, --human-readable — размер файловых систем выводится в понятном человеку виде и переводится в мегабайты/гигабайты вместо отображения этих данных в байтах;

– -H, --si — делает то же, что и параметр -h, но килобайт считается равным 1000 байт, а не 1024;

– -i, --inodes — вместо информации о блоках выдаётся информация об использовании индексных дескрипторов в файловой системе;

– -k, --kilobytes — при выводе устанавливает размер блока в 1024 байт;

– -l, --local — выводит только данные о локальных файловых системах;

– -m, --megabytes — при выводе устанавливает размер блока в мегабайт;

– --no-sync — не делать системный вызов sync перед получением данных об использовании дискового пространства (быстрый запуск);

– --sync — делать системный вызов sync перед получением данных об использовании дискового пространства;

– -t <тип ФС>, --type=<тип ФС> — показывать только файловые системы с указанным <тип ФС>;

– -T, --print-type — выдавать тип для каждой файловой системы;

– -x <тип ФС>, --exclude-type=<тип ФС> — не показывать файловые системы с заданным типом-файловой-системы. Можно задать несколько типов файловых систем, если использовать несколько параметров -x. По умолчанию никакие типы файловых систем не опускаются.

7.4.13. findmnt

Команда findmnt отображает список примонтированных файловых систем.

Формат команды:

```
findmnt [<параметры>] <раздел>|<точка монтирования>
```

Если указан раздел, показывает только точки монтирования, связанные с данным разделом. Если указана точка монтирования, выводятся только раздел, соответствующий данной точке монтирования. Команда позволяет получить информацию о разделе, точке монтирования, типе файловой системы и параметрах монтирования.

Параметры:

– -l — просмотр в виде списка;

– -t <тип ФС> — показать файловые системы только заданного типа.

7.4.14. du

Команда du выполняет оценку занимаемого дискового пространства. По умолчанию показывает объём дискового пространства, занимаемого каждым файлом и каталогом в текущем каталоге. Чтобы указать другой путь для работы, необходимо поместить его первым параметром.

Формат команды:

```
du [<параметры>] [<файл...>]
```

Отличающиеся параметры:

– -a, --all — показывать размеры для всех встретившихся файлов, а не только для каталогов;

– -b, --bytes — выдавать размеры в байтах вместо килобайтов;

– --block-size=<размер> — выдаёт размеры в блоках заданного размера (байт);

– -c, --total — выдавать общий итог по всем аргументам после того, как все аргументы будут обработаны. Это может быть использовано для выяснения суммарного использованного дискового пространства для всего списка заданных файлов и каталогов.

– -D, --dereference-args — раскрывать символьные ссылки, заданные в командной строке. Не оказывает влияния на остальные символьные ссылки. Это

полезно для поиска использованного дискового пространства в таких каталогах, как `/usr/tmp`, которые часто являются символьными ссылками.

– `-h`, `--human-readable` — размер файловых систем выводится в понятном человеку виде и переводится в мегабайты/гигабайты вместо отображения этих данных в байтах;

– `-H`, `--si` — делает то же, что и параметр `-h`, но килобайт считается равным 1000 байт, а не 1024;

– `-k`, `--kilobytes` — при выводе устанавливает размер блока в 1024 байт;

– `-m`, `--megabytes` — при выводе устанавливает размер блока в мегабайт;

– `-l`, `--local` — выводит только данные о локальных файловых системах;

– `-L`, `--dereference` — раскрывать все символьные ссылки;

– `--exclude=<шаблон>` — при рекурсивном выполнении пропускать каталоги или файлы, чьи имена совпадают с заданным шаблоном;

– `--max-depth=n` — выдавать общий итог для каталога (или файла, если задан параметр `-a`), только если он находится не более чем на `n` уровней глубины ниже заданного в командной строке аргумента; `--max-depth=0` означает то же самое, что и параметр `-s`;

– `-s`, `--summarize` — выдавать только суммарный итог для каждого аргумента;

– `-S`, `--separate-dirs` — выдавать отдельно размер каждого каталога, не включая размеры подкаталогов;

– `-x`, `--one-file-system` — пропускать каталоги, находящиеся не на той же файловой системе, что и обрабатываемый аргумент командной строки;

– `-X <файл>`, `--exclude-from=<файл>` — выполняет те же действия, что и параметр `--exclude`, за исключением того, что шаблоны берутся из указанного файла. Шаблоны перечисляются по одному на строку. Если файл задан как «-», то шаблоны читаются из стандартного ввода.

Примеры.

Размер файла:

```
$ du file.txt
1034
```

Размер файла в понятном человеку формате:

```
$ du -h archive.tar.gz
145M
```

Размер каждого подкаталога в каталоге /var/log:

```
# du -sh /var/log/*
8,0K   /var/log/abrt.log
16K    /var/log/aide
8,0K   /var/log/anaconda.ifcfg.log
32K    /var/log/anaconda.log
100K   /var/log/anaconda.program.log
264K   /var/log/anaconda.storage.log
84K    /var/log/anaconda.syslog
56K    /var/log/anaconda.xlog
108K   /var/log/anaconda.yum.log
...
```

Сортировать вывод команды du можно следующим образом:

```
# du -sh /var/log/* | sort -h
```

Параметр -h команды sort учитывает размерность каждого каталога (килобайты, мегабайты и т. п.).

7.4.15. eject

Команда eject программно извлекает лоток привода компакт-диска без необходимости нажимать на кнопку выдвижения лотка.

7.5. Утилиты работы с текстовыми файлами

В данном разделе перечислены утилиты для обработки, чтения и редактирования текстовых файлов.

7.5.1. cat

Утилита cat предназначена для вывода содержимого файла на терминал. Подразумевается, что файл текстовый, но утилита отображает и бинарные файлы. Утилита cat не форматирует исходные данные, предоставляя администратору их в исходном виде, и если исходный файл не помещается в окне терминала, то администратор не сможет программой просмотреть весь файл. Для просмотра файлов, не помещающихся в окне терминала, необходимо использовать команды less или more.

Формат команды:

```
cat [<параметры>] [<файл...>]
```

Параметры:

`-b` Нумеруются непустые строки файла.

`-s` Нумеруются все строки файла, поле номера отделяется от текста символом табуляции.

`-v` Визуализация непечатных символов.

С параметром `-v` можно использовать следующие дополнительные параметры:

`-t` Визуализация символов табуляции в виде `^I`.

`-e` Визуализация символов перевода строки в виде `$` (строка при этом всё равно переводится).

Если параметр `-v` не указан, то параметры `-t` и `-e` игнорируются.

Примеры.

Вывести содержимое файла `/proc/loadavg`:

```
$ cat /proc/loadavg
1.03 1.06 1.06 2/352 7814
```

Объединить файл `file1` с `file2` и записать в `file3`:

```
$ cat file1 file2 > file3
```

Объединить все файлы в каталоге в один большой файл:

```
$ cat * > bigfile
```

Быстро создать файл и добавить в него несколько строк текста можно при помощи команды `cat`. Для этого выполните команду с перенаправлением потока вывода в файл и начните вводить текстовую информацию. Завершить ввод информации нужно сочетанием клавиш [Ctrl+D]. Пример:

```
$ cat > newfile
line 1
line 2
line 3
^D
$ cat newfile
line 1
line 2
line 3
```

7.5.2. less

Утилита less предназначена для листания файлов, содержимое которых не помещается на один экран. Используется в конвейере для чтения длинного вывода команды. Листание производится клавишами клавиатуры [вверх], [вниз], [PageUp], [PageDown]. Выход из программы — клавиша [q].

Формат команды:

```
less [<параметры>] [<файл...>]
```

Примеры.

Чтение файла:

```
$ less file
```

Чтение вывода команды:

```
$ ls /etc | less
```

7.5.3. more

Утилита предназначена для простого листания документов. Аналогична less, однако при этом менее функциональна и может листать документ только сверху вниз. Для листания используется клавиша [пробел] или [Enter].

Формат команды:

```
more [<параметры>] [<файл...>]
```

Параметры:

-<N> Задается число, используемое в качестве размера окна (в строках), по умолчанию — 22.

+<N> Задается номер строки, с которой начинается вывод.

-d Вывод сообщения после каждого экрана.

-c Вывод каждого нового экрана с предварительной очисткой экрана.

-l Если не задан этот параметр, команда more останавливается после любой строки, содержащей [Ctrl-L], до тех пор, пока экран не заполнится до конца.

-p Вывод каждого нового экрана с верхней строки без предварительной очистки экрана.

-s Отображение нескольких пустых строк как одной.

+/шаблон Просмотр текста, начиная за две строки до той строки, в которой содержится шаблон, заданный регулярным выражением.

7.5.4. pr

Команда pr форматирует и выдает файлы на стандартный вывод. По умолчанию выдача разбивается на страницы, каждая из которых содержит в заголовке свой номер, дату, время и имя файла. При выводе в несколько колонок ширину строки можно задать, по умолчанию она равна 72.

Формат команды:

```
pr [<параметры>] [<файл...>]
```

Параметры:

+N Начать печать со страницы N (по умолчанию с 1).

-N Печать в N колонок (по умолчанию 1).

-m Слияние и печать всех файлов одновременно, по одному в колонке; максимальное число файлов — 8; несовместима с параметром -N.

-n[<символ>][<число>] Нумерация строк; номер занимает <число>+1 первых позиций каждой колонки при обычном выводе или каждой строки при выводе с параметром -m; если задан любой нецифровой символ, то он присоединяется к номеру строки, отделяя её от последующего текста.

-w<ширина> Установка ширины строки (по умолчанию — 72 символа); действует только при печати в несколько колонок.

-l<длина> Установка длины страницы (по умолчанию 66).

7.5.5. head

Команда head выводит в стандартный вывод первые строки файла (по умолчанию — 10 строк).

Формат команды:

```
head [<параметры>] [<файл...>]
```

Параметры:

-<N> или -n <N> Выводит первые N строк.

-c <размер>[b|k|m] Выводит последние <размер> байт файла, может иметь окончание: b — блоки по 512 байт, k — килобайт, m — мегабайт.

7.5.6. tail

Команда tail выводит в стандартный вывод последние строки файла (по умолчанию — 10 строк).

Формат команды:

```
tail [<параметры>] [<файл...>]
```

Параметры:

`-<N>` или `-n <N>` Выводит последние `N` строк.

`-c <размер>[b|k|m]` Выводит последние `<размер>` байт файла, может иметь окончание: `b` — блоки по 512 байт, `k` — килобайт, `m` — мегабайт.

`-f` Переходит в режим интерактивного чтения: `tail` не завершает работу, а продолжает выводит на экран новые сообщения, которые появились в файле, данный режим удобен для чтения файлов журнала.

7.5.7. `echo`

Команда выводит на терминал какое-либо сообщение. Удобна для вывода информационных сообщений в терминал или записи информации в файл.

Формат команды:

```
echo [-e] <текст...>
```

Параметр `-e` включает интерпретацию специальных символов, например, «\n».

7.5.8. `printf`

Выводит форматированные строки, аналогична функции `printf` из языка Си.

Формат команды:

```
printf <формат> [<значение...>]
```

7.5.9. `diff`

`diff` — утилита сравнения файлов, выводящая разницу между двумя файлами. Эта программа выводит построчно изменения, сделанные в файле (для текстовых файлов). Современные реализации поддерживают также двоичные файлы. Вывод утилиты называется «diff», или, что более распространено, патч, так как он может быть применён с программой `patch`.

Формат команды:

```
diff [<параметры>] <первый файл> <второй файл>
```

Параметры.

`-a` Считать все файлы текстовыми и сравнивать их построчно, даже если они не выглядят текстовыми.

`-b` Игнорировать изменения в количестве пробелов, табуляций и т. п.

`-B` Игнорировать изменения, касающиеся только вставки или удаления пустых строк.

`--brief` Извещать только о самом факте различия файлов, без каких-либо подробностей.

`-c` Использовать контекстный формат вывода.

`-C <количество строк> или --context[=<количество строк>]` Использовать контекстный формат вывода, показывая заданное количество строк контекста, или три строки, если это количество не задано. Для корректной работы программе `patch` обычно нужно не менее двух строк контекста.

`--changed-group-format=<формат>` Использовать заданный формат для вывода группы строк, содержащей различающиеся строки из обоих файлов в формате если-то-иначе.

`-d` Включает алгоритм поиска минимального набора изменений. Применение данного параметра делает работу `diff` более медленной (иногда очень медленной).

`-D <имя>` Показывает объединенные изменения в файлах, выводя их в формате если-то-иначе, с использованием директивы препроцессора `#define имя`.

`-e или --ed` Создаёт вывод в форме сценария для `ed` (см. раздел 7.5.27).

`--exclude=<шаблон>` При сравнении каталогов игнорирует файлы и подкаталоги, чьи имена совпадают с шаблоном.

`--exclude-from=<файл>` При сравнении каталогов игнорирует файлы и подкаталоги, чьи имена совпадают с шаблонами, находящимся в файле.

`--expand-tabs` При выводе заменяет табуляцию пробелами для сохранения выравнивания во входных файлах.

`-f` Делает вывод похожим на сценарий для `ed`, но изменения показываются в том порядке, в котором они встречаются в файле.

`-F <регулярное выражение>` В контекстном и унифицированном формате, для каждой порции различий, показывать несколько строк, предшествующих этой порции, которые совпадают с регулярным выражением (подробное описание регулярных выражений приведено в разделе 7.6).

`--forward-ed` Делает вывод похожим на сценарий для `ed`, но изменения показываются в том порядке, в котором они встречаются в файле.

`-H` Использовать эвристики для быстрой обработки больших файлов, которые имеют несколько маленьких изменений, разбросанных по файлу.

`--horizon-lines=<строки>` Не отбрасывать последние `<строки>` общих для обоих файлов перед и первые `<строки>` общих для обоих файлов строк после.

`-i` Игнорировать изменения в регистре символов; считать буквы верхнего и нижнего регистров (строчные и прописные) эквивалентными.

`-I <регулярное выражение>` Игнорировать изменения, которые касаются только вставки или удаления строк, совпадающих с регулярным выражением (подробное описание регулярных выражений приведено в разделе 7.6).

`--ifdef=<имя>` Показывает объединенные изменения в файлах, выводя их в формате если-то-иначе, с использованием директивы препроцессора `#define <имя>`.

`--ignore-all-space` Игнорировать изменения в количестве пробелов, табуляций и т. п.

`--ignore-blank-lines` Игнорировать изменения, касающиеся только вставки или удаления пустых строк.

`--ignore-case` Игнорировать изменения в регистре символов; считать буквы верхнего и нижнего регистров (строчные и прописные) эквивалентными.

`--ignore-matching-lines=<регулярное выражение>` Игнорировать изменения, касающиеся только вставки и удаления строк, совпадающих с регулярным выражением (подробное описание регулярных выражений приведено в разделе 7.6).

`--ignore-space-change` Игнорировать изменение количества пробелов, табуляций и т. п.

`--initial-tab` Выводить перед строкой текста в нормальном или контекстном формате табуляцию вместо пробела. За счет выравнивания по границам табуляции получается нормальный вид строки.

`-l` Передавать результат команде `pr` для разбития его на страницы.

`-L <метка>, --label=<метка>` Использовать метку вместо имени файла в заголовке контекстного и унифицированного формата.

`--left-column` Выводить только левую колонку для двух общих (для обоих файлов) строк при двухстороннем формате.

`--line-format=<формат>` Использовать заданный формат для вывода всех входных строк в формате если-то-иначе.

`--minimal` Включает алгоритм поиска минимального набора изменений. Применение данного параметра делает работу `diff` более медленной (иногда очень медленной).

`-n, --rcs` Вывод в формате RCS-diff; как и в параметре `-f` за исключением того, что каждая команда задает количество затронутых строк.

`-N, --new-file` При сравнении каталогов, если файл найден только в одном каталоге, то считать, что он существует и в другом каталоге, но является пустым.

`--new-group-format=<формат>` Использовать заданный формат для вывода группы строк, которая берется только из второго файла в формате если-то-иначе.

`--new-line-format=<формат>` Использовать заданный формат для вывода строки, которая берется только из второго файла в формате если-то-иначе.

`--old-group-format=<формат>` Использовать заданный формат для вывода группы строк, которая берется только из первого файла в формате если-то-иначе.

`--old-line-format=<формат>` Использовать заданный формат для вывода строки, которая берется только из первого файла в формате если-то-иначе.

`-p, --show-c-function` Показывать, внутри каких функций языка C происходит каждое изменение.

`-P` При сравнении каталогов, если файл существует только во втором каталоге, то считать, что он есть и в другом каталоге, но только пустой.

`--paginate` Передавать результат команде `pr` для разбития его на страницы.

`-q` Извещать только о самом факте различия файлов, без каких-либо подробностей.

`-r, --recursive` При сравнении каталогов производить рекурсивное сравнение всех найденных подкаталогов.

`--report-identical-files, -s` Сообщать, что два файла являются одинаковыми.

`-S <файл>` При сравнении каталогов начинать с файла `<файл>`. Данный параметр используется для продолжения прерванного процесса сравнения.

`--sdiff-merge-assist` Выдавать дополнительную информацию, чтобы помочь программе `sdiff`. `sdiff` использует данный параметр, когда он запускает `diff`.

`--show-function-line=<регулярное выражение>` В контекстном и унифицированном формате, для каждой порции различий, показывать несколько строк, предшествующих этому изменению, которые совпадают с регулярным_выражением (подробное описание регулярных выражений приведено в разделе 7.6).

`--side-by-side` Использовать двухсторонний формат вывода.

`--speed-large-files` Использовать эвристики для быстрой обработки больших файлов, которые содержат несколько небольших изменений, разбросанных по файлу.

`--starting-file=<файл>` При сравнении каталогов начинать с файла файл. Данный параметр используется для продолжения прерванного процесса сравнения.

`--suppress-common-lines` Не выводить общие для обоих файлов строки в двухстороннем формате.

`-t` При выводе заменяет табуляцию пробелами для сохранения выравнивания.

`-T` Выводить перед строкой текста в нормальном или контекстном формате табуляцию вместо пробелов. За счет выравнивания по границам табуляции получается нормальный вид строки.

`--text` Считать все файлы текстовыми и сверять их построчно, даже если они не выглядят, как текстовые.

`-u` Использовать унифицированный формат вывода.

`--unchanged-group-format=<формат>` Использовать заданный формат для вывода группы общих для обоих файлов строк, которые берутся из обоих файлов в формате если-то-иначе.

`--unchanged-line-format=<формат>` Использовать заданный формат для вывода общей строки (для обоих файлов) в формате если-то-иначе.

`--unidirectional-new-file` При сравнении каталогов, если файл существует только во втором каталоге, то считать, что он есть и в другом каталоге, но только пустой.

`-U <количество строк> или --unified[=<количество строк>]` Использовать унифицированный формат вывода, показывая количество_строк содержимого или три строки, если это количество не задано. Для корректной работы программе `patch` обычно нужно не менее двух строк содержимого.

`-w` Игнорировать пробелы и табуляции при сравнении строк.

-W <ширина> или --width=<ширина> Использовать при выводе в двухстороннем формате колонки заданной ширины.

-x <шаблон> При сравнении каталогов игнорирует файлы и подкаталоги, чьи имена совпадают с шаблоном.

-X <файл> При сравнении каталогов, игнорирует файлы и подкаталоги, чьи имена совпадают с шаблонами, находящимися в файле.

-y Использовать двухсторонний формат.

Примеры.

Создания патча исходного кода, пригодного для программы patch:

```
$ diff -ruN source.c source.c.old
```

Создание патча на основе двух каталогов, включая новые файлы:

```
$ diff -Naur source-dir source-dir.old
```

7.5.10. cmp

Команда cmp сравнивает два файла, и если они различаются, сообщает о первом байте и строке, где было обнаружено различие.

Формат команды:

```
cmd <параметры> <файл1> [<файл2>]
```

Код возврата:

– 0 — различий не найдено;

– 1 — есть различия;

– 2 — произошла ошибка.

Параметры:

-c Печатает различающиеся символы. Отображает контрольные символы символом «^» и буквой алфавита, а также предваряет символы с установленным высшим битом символом «-M» (обозначающим «мета»).

--ignore-initial=<байт> Игнорирует все различия в первых <байт> байтах входных файлов. Обращается с файлами меньшими по размеру, чем <байт>, как с пустыми.

-l Печатает смещение (десятичное) и значение (восьмеричное) всех различающихся байтов.

--print-char Печатает все различающие символы. Отображает контрольные символы символом «^» и буквой алфавита, а также предваряет символы с установленным высшим битом символом «-M» (обозначающим «мета»).

`--quite` или `-s` или `--silent` Ничего не печатает; только возвращает выходной статус, показывающий отличаются ли файлы.

`--verbose` Печатает смещение (десятичное) и значение (восьмеричное) всех различающихся байтов.

7.5.11. `comm`

Команда `comm` читает два файла (их содержимое должно быть отсортировано в лексикографическом порядке) и выводит результаты в три колонки:

1) строки, входящие только в `<файл1>;`

2) строки, входящие только в `<файл2>;`

3) строки, входящие в оба файла.

Формат команды:

```
comm [<параметры>] <файл1> <файл2>
```

Параметр `-N` подавляет вывод соответствующей колонки `N`.

7.5.12. `cut`

Команда `cut` выводит выбранные части строк (столбцы) каждого заданного файла. Вместо файла может принимать данные на стандартный ввод, что делает её удобной для обработки вывода других команд.

Формат команды:

```
cat [<параметры>] [<файл...>]
```

Параметры:

`-b, --bytes=<список>` Выводит только байты из позиций, указанных в списке. Символы табуляции и BackSpace трактуются подобно другим символам и занимают один байт.

`-c, --characters=<список>` Выводит только символы из позиций, указанных в списке.

`-d, --delimiter=<разделитель>` Задаёт разделитель колонок (полей) входного файла вместо символа табуляции. Применяется совместно с параметром `-f`.

`-f, --fields=<список>` Выводит только столбцы (поля), перечисленные в списке. По умолчанию столбцы разделяются символами табуляции. Если не задан параметр `-s`, выводит любые строки, которые не содержат символ-разделитель.

`-s, --only-delimited` Не выводит строки, в которых отсутствует разделитель полей. Применяется совместно с параметром `-f`.

`-n` Не разбивает на части многобайтовые символы (игнорируется).

`--output-delimiter=<разделитель>` Разделяет указанным разделителем поля выходного потока. Применяется совместно с параметром `-f`. По умолчанию используется разделитель полей входного файла (потока).

Одновременно можно использовать только один из параметров `-b`, `-c` или `-f`.

Вы не ограничены выводом одного столбца, т. е. в списках могут содержаться один или более номеров или диапазонов, разделенных запятыми (пример: `1-3,5,6,8,18`). Каждый диапазон представляет собой два числа, разделенных дефисом (`5-12`). Байты, символы и поля нумеруются, начиная с 1. Могут задаваться неполные диапазоны. Так, если опустить нижнюю границу (`-19`), то будет использоваться диапазон (`1-19`) включительно. Если опустить верхнюю границу (`3-`), то диапазон будет ограничиваться концом строки или последним полем.

П р и м е ч а н и е. Скорость работы программа `cut` значительно уступает языку `awk`, для больших объёмов входных данных рекомендуется использовать `awk` (см. 7.5.29). Кроме того, `awk` может изменить порядок полей и их форматирование, применять фильтрацию и т. п.

Примеры.

Вывести имена всех пользователей:

```
$ cut -d: -f1 /etc/passwd
```

Вывести имена файлов, их права доступа и владельца:

```
$ ls -l | tr -s ' ' ' ' | cut -d' ' -f1,3,9-
```

7.5.13. `paste`

Выводит на стандартный вывод строки, состоящие из соответствующих строк каждого файла, разделенных символом табуляции.

Формат команды:

```
paste [<параметры>] [<файл...>]
```

Параметры:

– `-d, --delimiters=<список>` — использовать символы из списка вместо символа табуляции;

– `-s, --serial` — вставлять один файл за раз, а не параллельно с другим.

7.5.14. wc

Печатает число переводов строк, слов и байт для каждого файла и итоговую сумму, если было задано несколько файлов.

Формат команды:

```
wc [<параметры>] [<файл...>]
```

Параметры.

`-c, --bytes` Вывести количество байт.

`-m, --chars` Вывести количество символов.

`-l, --lines` Вывести количество строк.

`-L, --max-line-length` Вывести количество символов в самой длинной строке.

`-w, --words` Вывести количество слов.

7.5.15. uniq

Удаляет все кроме одной повторяющиеся строки из стандартного ввода и печатает их на стандартный вывод.

Формат команды:

```
uniq [<параметры>] [<входной файл>[<выходной файл>]]
```

Параметры.

`-c, --count` Выводить число повторов в начале каждой строки.

`-d, --repeated` Выводить только повторяющиеся строки.

`-D, --all-repeated[=none|prepend|separate]` Печатать все повторяющиеся строки.

`-f, --skip-fields=N` Не сравнивать первые `N` полей.

`-i, --ignore-case` Игнорировать регистр при сравнении.

`-s, --skip-chars=N` Не сравнивать первые `N` знаков.

`-u, --unique` Выводить только неповторяющиеся строки.

`-w, --check-chars=N` Сравнивать только первые N символов в строке.

П р и м е ч а н и е. Входные данные должны быть отсортированы (например, командой `sort`).

Примеры.

Подсчитать количество правил для каждого домена КСЗ:

```
$ sesearch -A|awk '{print $2}'|sort|uniq -c|sort -n
```

То же, но с использованием `awk` (данный вариант быстрее предыдущего):

```
$ sesearch -A|awk '{d[$2]++}END{for(i in d)print d[i],
i}'|sort -n
```

7.5.16. `sort`

Команда `sort` сортирует строки текстовых файлов.

Формат команды:

`sort [<параметры>] [<файл...>]`

Параметры.

`-b, --ignore-leading-blanks` Игнорировать пробелы в начале сортируемых полей или начале ключей.

`-d, --dictionary-order` Воспринимать в составе ключей лишь буквы (латинского алфавита), цифры и пробелы, игнорируя все прочие символы.

`-f, --ignore-case` Во время сортировки преобразует строчные (маленькие) в соответствующие прописные (большие) буквы, т. е. выполняется сортировка нечувствительная к регистру символов.

`-g, --general-numeric-sort` Выполнять сравнение в соответствии с общим числовым значением (используют совместно с параметром `-b`). Это численная сортировка, при которой дополнительно распознаётся экспоненциальное представление чисел (например, 9.1019e7).

`-i, --ignore-nonprinting` В ключах рассматриваются только печатаемые символы, а остальные игнорируются.

`-M, --month-sort` Выполнять сравнение по трёх-символьным сокращениям англоязычных названий месяцев.

`-n, --numeric-sort` Числовая сортировка, т. е. сравнение ведётся по числовому значению (используют совместно с параметром `-b`).

`-r, --reverse` Сортировка выполняется в обратном порядке (по убыванию).

`-c, --check` Проверяет сортировался ли указанный файл. Если да, то не выполняет сортировку, иначе выводит сообщение об ошибке.

`-k, --key=<позиция...>` Выбирает ключ сортировки, начиная с первой указанной позиции и заканчивая последней указанной позицией. Номера полей и смещения символов указываются, начиная с 1.

`-m, --merge` Объединяет ранее отсортированные файлы, которые не сортируются повторно.

`-o, --output=<файл>` Выводит результат в указанный файл вместо стандартного вывода.

`-s, --stable` Стабилизирует сортировку, не выполняя сравнения последней пересортировки.

`-S, --buffer-size=<размер>` Под основной буфер в памяти использует область указанного размера.

`-t, --field-separator=<символ>` Использовать `<символ>` в качестве разделителя полей.

`-T, --temporary-directory=<каталог>` Использует указанный `<каталог>` для временных файлов, игнорируя переменную окружения `TMPDIR` или `/tmp`; составные параметры могут указывать на различные каталоги.

`-u, --unique` Уникальная сортировка: игнорирует повторяющиеся строки. Обычно применяют с параметром `-c` для проверки сортированных файлов с целью прерывания выполнения, если встретится несколько одинаковых строк подряд; без `-c` выводится только первая строка из одинаковых.

`-z, --zero-terminated` Вместо символа новой строки, завершает строки двоичным 0.

Синтаксис параметра `-k, --key=<позиция...>` означает следующее:

Позиция указывается в формате `F[.C][OPTS]`, где `F` является порядковым номером поля, а `C` — позицией символа в этом поле. `OPTS` представляет собой одну или более одиночных букв, которые означают рассмотренные выше параметры и их действие перекрывает действие глобальных параметров для этого ключа. Если ни один ключ не задан, в качестве ключа используется вся строка. Ключ сортировки — это часть строки, которая рассматривается при сортировке, вместо того чтобы рассматривалась вся строка. Таким образом, команда:

```
$ sort -k1.3
```

осуществляет сортировку по первому полю, начиная с его третьего символа. Команда

```
$ sort -k4.6,7n
```

означает сортировку с 6-го символа четвёртого поля до 1-го символа седьмого поля. Сортировка ведётся по числовому значению. Допускается задание нескольких параметров `-k` для того, чтобы определить несколько ключей, которые будут использованы последовательно в том порядке, в котором они указаны в командной строке.

Величина размера памяти, отводимой для работы команды `sort` может быть задана со следующими суффиксами: «%» процент от общей памяти; b — в байтах; K — 1024 байт (по умолчанию); и соответственно для M, G, T, P, E, Z, Y.

Если в командной строке не указан никакой файл или вместо имени стоит дефис, считывается стандартный ввод (с клавиатуры).

П р и м е ч а н и е. На сортировку влияют установки локали. Установите переменную окружения `LC_ALL=C`, чтобы получить обычную сортировку в том порядке, который использует действительные значения байт.

Например.

Определить, какой подкаталог в каталоге `/usr` занимает больше всего места. Команда `sort -h` выполняет сортировку с учётом размерности величины (гигабайты, мегабайты и т. п.).

```
du -sh /usr/* | sort -h
```

7.5.17. `join`

Команда `join` объединяет строки двух файлов в общее поле. Для каждой пары строк ввода с одинаковыми общими полями, записывает строку в стандартный вывод. По умолчанию общее поле считается первым, поля разделяются пробельными знаками.

Формат команды:

```
join [<параметры>] <файл1> <файл2>
```

Параметры.

`-a N` Печатать не имеющие пары строки из файла с заданным номером (1 или 2).

`-e <строка>` Замещать при выводе пустые строки указанной строкой.

`-i, --ignore-case` Игнорировать регистр букв при сравнении полей.

`-j <поле>` Эквивалентно `-1 <поле> -2 <поле>`.

`-o <формат>` Выводить в соответствии с форматом.

`-t <символ>` Использовать символ как разделитель поля ввода и поля вывода.

`-v <номер>` Как `-a <номер>`, но не печатать имеющие пары строки.

`-1 <поле>` Считать общим заданное поле `<файл1>`.

`-2 <поле>` Считать общим заданное поле `<файл2>`.

Если не задан -t <символ>, используются пробельные символы для разделения. Каждое поле является порядковым номером, начиная с 1. <формат> — это одно или несколько разделяемых запятыми или пробельными знаками описаний формата в виде <номер файла>.<поле> или 0. По умолчанию <формат> выводит общее поле, остальные поля из <файл1> и остальные поля из <файл2>, разделенные <символ>.

П р и м е ч а н и е. Оба файла должны быть отсортированы по общим полям (см. 7.5.16).

7.5.18. split

Команда split читает файл и записывает его заданными порциями (по умолчанию порция оставляет 1000 строк) в набор выходных файлов.

Формат команды:

split [<параметры>] [<файл...> [<префикс>]]

Имя первого выходного файла получается из префикса путём добавлением к нему суффикса aa, ab, и далее в лексикографическом порядке вплоть до zz. Если префикс не указан, используется x. Например, задан префикс out, тогда выходными файлами будут outaa, outab, outac и т. п.

Параметры.

-a, --suffix-length=N Суффикс длины N (по умолчанию 2).

-b, --bytes=N Делить файл на куски по N байт.

-C, --line-bytes=N Делить файл на куски по N байт, сохраняя целостность строк.

-d, --numeric-suffixes Использовать числовой суффикс вместо символьного.

-l, --lines=N Делить файл на N строк.

7.5.19. tee

Команда tee читает стандартный ввод и записывает его содержимое в файл, при этом передавая стандартный ввод далее по конвейеру.

Формат команды:

tee [-a] [<файл...>]

Параметр -a означает, что команда будет добавлять данные в файл, а не перезаписывать его.

Данная команда обычно используется в конвейере совместно с другими командами, когда нужно сохранить содержимое конвейера на каком-либо этапе в файл или выполнить распараллеливание потоков вывода с использованием именованных каналов.

7.5.20. tr

Выполняет преобразование, подстановку (замену), сокращение и/или удаление символов, поступающих со стандартного ввода, записывая результат на стандартное устройство вывода. Она часто применяется для удаления управляющих символов из файла или преобразования регистра символов. Как правило, команде tr передаются две строки (набора) символов: первый набор содержит искомые символы, а второй — те, на которые их следует заменить. При запуске команды устанавливается соответствие между символами обоих наборов, а затем начинается преобразование.

Формат команды:

```
tr [<параметры>] <строка1> <строка2>
```

Параметры.

`-c, --complement` Замещает первый набор символов <строка1> его дополнением (всеми символами, отсутствующими в <строка1>).

`-d, --delete` Удаляет все символы, которые перечислены в наборе <строка1> без преобразования.

`-s, --squeeze-repeats` Заменяет последовательность повторяющихся символов в наборе <строка1> на один такой символ (т. е. удаляет все повторяющиеся символы, кроме первого).

`-t, --truncate-set1` Ограничивает (делает обрезание) набор <строка1>, если он длиннее набора <строка2>.

<строка1> и <строка2> — это строки символов. При указании команде tr содержимого наборов <строка1> или <строка2> используются только диапазоны и последовательности символов либо отдельные символы:

– \NNN — восьмеричное число, состоящее из трёх цифр NNN и представляющее любой действительный символ в коде ASCII;

– \\ — символ обратной косой черты (обратный слеш);

– \a — символ — звонок (BEL);

– \b — символ возврата на одну позицию (BackSpace);

– \f — символ прокрутки страницы;

– \n — символ новой строки;

– \r — символ возврата каретки (return);

– \t — символ горизонтальной табуляции (tab);

– \v — символ вертикальной табуляции;

– <символ1>-<символ2> — все символы из диапазона от <символ1> до <символ2> включительно;

– [<символ>*] — указанный в наборе <строка2>, выполняет копирование СИМВОЛА в количестве равном длине набора <строка1>;

– [<символ>*N] — копировать заданное количество раз (N) <символ>; если N начинается с 0 — это означает, что оно задано в восьмеричной форме.

– [:alnum:] — все буквы и цифры;

– [:alpha:] — все буквы;

– [:blank:] — все горизонтальные символы пробела (пробел, табуляция);

– [:cntrl:] — все управляющие символы;

– [:digit:] — все цифры;

– [:graph:] — все печатные символы, исключая пробел;

– [:lower:] — все строчные буквы (нижнего регистра);

– [:print:] — все печатные символы, включая пробел;

– [:punct:] — все знаки пунктуации;

– [:space:] — все горизонтальные или вертикальные пробелы;

– [:upper:] — все прописные буквы (ВЕРХНЕГО регистра);

– [:xdigit:] — все шестнадцатеричные цифры;

– [=<символ>=] — все символы, которые эквивалентны <символ>.

Если заданы оба набора символов и не указан параметр -d, команда tr преобразует каждый символ <строка1> в соответствующий символ <строка2>. Параметр -t используется только при преобразованиях. В случае необходимости, <строка2> расширяется до длины <строка1>, повторением её последнего символа. Лишние символы <строка2> игнорируются. Гарантированно могут расширяться в порядке возрастания только диапазоны [:lower:] и [:upper:]; используя их во время преобразования в <строка2>, они должны быть заданы только в паре, чтобы определить вариант преобразования. Параметр -s использует <строка1> в том случае, если не выполняется преобразование или удаление; иначе выполня-

ется сокращение повторяющихся символов с использованием <строка2>, а после этого выполняется преобразование или удаление.

Следует помнить, что при замене строки или диапазона символов одним символом, этот символ не указывается в квадратных скобках ([]), хотя в некоторых системах допускается применение квадратных скобок, причём для указания, например, символа новой строки можно воспользоваться шаблоном ["\012"] или "\012". Команда tr не предъявляет строгих требований к виду кавычек.

Подобно большинству системных команд, tr восприимчива к специальным символам. Поэтому если необходимо выполнить сопоставление с одним из таких символов, его следует предварительно отделить обратной косой чертой, например, \{ — для отмены специального значения фигурной скобки.

7.5.21. fold

Утилита fold выполняет перенос длинных строк так, чтобы они уместились на экране терминала.

Формат команды:

```
fold [<параметры>] [<файл>]
```

Параметры.

-b, –bytes Считать длину строки в байтах.

-c, –characters Считать длину строки в символах.

-w, –width=N Разбивать текст по ширине N столбцов вместо 80.

Пример.

Команду можно использовать для разбиения строки на символы:

```
$ echo "Hello world" | fold -w1
H
e
l
l
o

w
o
r
l
```

```
d
```

Исходя из примера выше, можно подсчитать частоту использования каждого символа алфавита в тексте:

```
$ echo ”Hello world”|fold -w1|sort|uniq -c|sort -n
      1
      1 d
      1 e
      1 H
      1 r
      1 w
      2 o
      3 l
```

7.5.22. iconv

Программа iconv изменяет кодировку символов файла. Результат выводится на стандартное устройство вывода, если не определен файл вывода параметром --output.

Формат команды:

```
iconv -f <исходная кодировка> -t <целевая кодировка>
<входной файл>
```

Параметры.

--from-code, -f <исходная кодировка> Преобразование из кодировки <исходная кодировка>.

--to-code, -t <целевая кодировка> Преобразование в кодировку <целевая кодировка>.

--list Выдать список известных кодировок.

--output, -o <файл> Определить файл вывода (вместо стандартного вывода).

7.5.23. enca

Программа enca определяет кодировку и конвертирует файлы.

Формат команды:

```
enca [-L <язык>] [<параметры>]... [<файл>]...
```

Параметры.

`-L` Использовать язык для автоматического определения кодировки.

`-c` Автоматически конвертировать файл в кодировку по умолчанию.

7.5.24. `grep`

Команда `grep` выполняет поиск строк, соответствующих шаблону, заданному регулярным_выражением, в файлах или во входном потоке. Если команда задана без параметров, выводятся все найденные строки. Если имя файла не задано, команда выполняет поиск во входном потоке. Если задано несколько имен файлов или в составе имени файла использован символ «*», `grep` перед строкой выводит имя файла, которому эта строка принадлежит.

Формат команды:

```
grep [<параметры>] [<шаблон>] [<файл...>]
```

7.5.24.1. Параметры

Выбор типа регулярного выражения и его интерпретация.

`-E, --extended-regexp <шаблон>` Расширенное регулярное выражение.

`-F, --fixed-regexp <шаблон>` Строки фиксированной длины, разделённые символом новой строки.

`-G, --basic-regexp <шаблон>` Простое регулярное выражение.

`-P, --perl-regexp <шаблон>` Регулярное выражения языка Perl.

`-e, --regexp=<шаблон>` Использовать `<шаблон>` для поиска.

`-f, --file=<файл>` Брать `<шаблон>` из `<файл>`.

`-i, --ignore-case` Игнорировать различие регистра.

`-w, --word-regexp <шаблон>` Должен подходить ко всем словам.

`-x, --line-regexp <шаблон>` Должен подходить ко всей строке.

`-z, --null-data` Строки разделяются байтом с нулевым значением, а не символом конца строки.

Управление выводом.

`-m, --max-count=N` Остановиться после указанного `N` совпадений.

`-b, --byte-offset` Печатать вместе с выходными строками смещение в байтах.

`-n, --line-number` Печатать номер строки вместе с выходными строками.

`--line-buffered` Сбрасывать буфер после каждой строки.

`-H, --with-filename` Печатать имя файла для каждого совпадения.

`-h, --no-filename` Не начинать вывод с имени файла.

`--label=<метка>` Выводить `<метка>` в качестве имени файла для стандартного ввода.

`-o, --only-matching` Показывать только часть строки, совпадающей с шаблоном.

`-q, --quiet, --silent` Подавить весь обычный вывод.

`--binary-files=binary|text|without-match` Считать двоичный файл указанного типа: `binary` — считать, что файл двоичный, `text` — считать, что файл текстовый, `without-match` — ничего не предпринимать.

`-a, --text` То же что и `--binary-files=text`.

`-I` То же, что и `--binary-files=without-match`.

`-d, --directories=read|recurse|skip` Как обрабатывать каталоги: `read` (читать), `recurse` (рекурсивно), или `skip` (пропускать).

`-D, --devices=read|skip` Как обрабатывать устройства, именованные каналы и сокеты: `read` (читать) или `skip` (пропустить).

`-R, -r, --recursive` То же, что и `--directories=recurse`.

`--include=<шаблон>` Обработать только файлы, подпадающие под `<шаблон>`.

`--exclude=<шаблон>` Пропустить файлы и каталоги, подпадающие под `<шаблон>`.

`--exclude-from=<файл>` Пропустить файлы, подпадающие под шаблон файлов из `<файл>`.

`--exclude-dir=<шаблон>` Каталоги, подпадающие под `<шаблон>`, будут пропущены.

`-L, --files-without-match` Печатать только имена файлов без совпадений.

`-l, --files-with-matches` Печатать только имена файлов с совпадениями.

`-c, --count` Печатать только количество совпадающих строк на файл.

`-T, --initial-tab` Выравнивать табуляцией (если нужно).

`-Z, --null` Печатать нулевой байт после имени файла.

Управление контекстом.

`-B, --before-context=N` Печатать `N` строк предшествующего контекста.

`-A, --after-context=N` Печатать `N` строк последующего контекста.

`-C, --context[=N]` Печатать `N` строк контекста.

`--color[=always|never|auto]` Использовать маркеры для различия совпадающих строк: `always` (всегда), `never` (никогда), или `auto` (автоматически, вывод в терминал будет цветным, а перенаправление потока вывода — нет).

`-U, --binary` Не удалять символы CR в конце строки.

`-u, --unix-byte-offsets` Выдавать смещение, как-будто нет CR-ов.

Разное.

`-s, --no-messages` Подавлять сообщения об ошибках.

`-v, --revert-match` Выбирать не подходящие строки.

`--mmap` При возможности, использовать ввод, спроецированный в память.

Команда `grep` допускает использование регулярных выражений. Подробное описание регулярных выражений приведено в разделе 7.6.

7.5.25. `egrep`

Аналог команды `grep -E`. Подробное описание см. в описании команды `grep`.

7.5.26. `fgrep`

Аналог команды `grep -F`. Подробное описание см. в описании команды `grep`.

7.5.27. `ed`

`ed` — интерактивный текстовый редактор.

Формат команды:

`ed [<параметры>] [<файл...>]`

Утилита `ed` производит редактирование текстовых файлов в соответствии с командами редактирования, задаваемыми пользователем в в командной строке. При вызове `ed` содержимое файла копируется в буфер (рабочую копию), и все редактирование происходит в буфере. Содержимое буфера копируется в редактируемый файл только по команде `w`.

При вводе в командной строке ошибочной команды `ed` печатает символ «?». Диагностику ошибки можно получить командой `H`.

Параметры/

`-s` Подавление печати количества символов при выполнении команд `e`, `r` и `w`, выдачу диагностики команд `e` и `q`.

`-p <приглашение>` Установка собственного текста приглашения.

Команды редактирования имеют формат:

`[<адрес1>[,<адрес2>]] команда`

Адрес определяет строки, к которым применяется команда или команды. Если заданы и первый, и второй адреса, то команда применяется к строкам от первого до второго адреса включительно. Если задан только первый адрес, то команда применяется к строкам, определяемым этим адресом. Если адреса не заданы, то команда применяется к текущей строке, если иное не оговорено в описании команды.

Адрес может задаваться:

– номером строки;

– регулярным выражением, заключенным в символы «/.../» или «?...?» (подробное описание регулярных выражений приведено в разделе 7.6);

– специальным символом «$», адресующим последнюю строку файла;

– специальным символом «.», адресующим текущую строку файла;

– специальным символом «%», эквивалентным адресу: «1,$»;

– специальным символом «;», эквивалентным адресу: «.,$».

При запуске `ed` текущей становится последняя строка текста. После выполнения любой команды текущей становится последняя строка, обработанная данной командой.

Изменить адрес текущей строки можно, введя одну из следующих команд:

– `<адрес>` — текущей становится команда, определяемая адресом;

– `[<адрес>]+[<число>]` или `[<адрес>]-[<число>]` — текущей становится команда, расположенная со смещением `<число>` от заданного адреса; если `<адрес>` не задан, предполагается текущий адрес; если `<число>` не задано, предполагается 1.

Если шаблон в адресе заключён в символы «/.../», поиск ведется от текущей строки вниз; «?...?» — поиск ведётся от текущей строки вверх. Если при поиске вниз достигнут конец текста, поиск продолжается с начала текста. Если при поиске вверх достигнуто начало текста, поиск продолжается с конца текста.

Если в адресных командах не указывается адрес, по умолчанию подразумевается «.» — текущая строка, если в описании команды не оговорено иное.

Далее в описании команд редактирования адресное выражение будет условно обозначаться символом «@».

Перечень доступных команд редактирования представлены в таблице 9.

Таблица 9 – Команды редактирования редактора `ed`

Команда	Описание
`H`	Вывод диагностики сделанной ошибки
`[@]p`	Вывод адресуемых строк на печать
`[@]=`	Вывод номера адресуемой строки; требуется только один адрес; по умолчанию здесь предполагается адрес $
`[@]a`	Переход в режим ввода текста, новый текст размещается после адресуемой строки. Команда требует только одного адреса в адресном выражении (по умолчанию — текущая строка). В режиме ввода вводимый текст добавляется в рабочую копию файла. Для выхода из режима ввода нужно ввести строку, состоящую из единственной точки
`[@]i`	То же, что и команда a, но новый текст размещается перед адресуемой строкой
`[@]c`	Переход в режим ввода текста, новый текст заменяет блок адресуемых строк
`[@]r [файл]`	Чтение текста из файла и вставка его после адресуемой строки. Если файл не задан, подразумевается текущий редактируемый файл. По умолчанию предполагается адрес $. В этой команде допустим адрес 0, он означает вставку перед первой строкой
`[@]d`	Удаление адресуемых строк
`[@]mадрес`	Перемещение блока адресуемых строк по адресу адрес, который не должен попадать в диапазон адресуемых строк
`[@]tадрес`	Копирование блока адресуемых строк по адресу адрес, который не должен попадать в диапазон адресуемых строк

Команда	Описание
[адрес]s/шаблон /текст[флаги]	Замена в адресуемых строках заданного шаблона заданным текстом. Шаблон задается регулярным выражением (подробное описание регулярных выражений приведено в разделе 7.6). В тексте может использоваться метасимвол «&» для обозначения заменяемого текста. Возможные флаги команды s: число — замена задаваемого числом вхождения шаблона (по умолчанию заменяется только первое вхождение); g — замена всех вхождений шаблона в строку
[@]w [файл]	Вывод адресуемых строк из буфера в файл. Если файл не задан, подразумевается текущий редактируемый файл. По умолчанию предполагается адрес 1,$, то есть все строки
e [файл]	Команда уничтожает содержимое буфера и читает в буфер заданный файл (по умолчанию — текущий редактируемый файл). Имя текущего редактируемого файла при этом не изменяется
q	Выход из редактора ed

7.5.28. sed

Команда sed — потоковый редактор. Потоковый редактор используется для выполнения основных преобразований с текстом на входном потоке (файл или ввод из конвейера). Тогда как некоторые схожие редакторы (такие как ed) допускают комплексное редактирование, sed работает совершая лишь одно действие над вводом за проход и следовательно более эффективен. Благодаря этому sed в состоянии фильтровать текст в конвейере, что особо выгодно отличает его от редакторов других типов.

Несмотря на то, что sed позволяет выполнять достаточно широкий спектр задач (см. примеры), зачастую лучше заменить его на ту утилиту, которая лучше подходит, так как сложные регулярные выражения в sed могут привести к падению производительности.

Формат команды:

```
sed [<параметры>] [<сценарий sed>] [<файл...>]
```

Если не указано ни одного файла, используется стандартный ввод. Результат работы `sed` направляется в выходной поток, если иное не задано в операторах редактирования.

Параметры:

`-n` Подавление печати всех просмотренных `sed` строк; выводятся только те строки, печать которых определена в командах редактирования.

`-i` Вместо отправки результата в стандартный вывод, осуществляет замену исходного файла.

`-r` Использовать расширенные регулярные выражения.

`-f <файл>` Чтение скрипта не из командной строки, а из файла.

7.5.28.1. Операторы редактирования

Операторы редактирования имеют формат:

```
[адрес1[,адрес2]] команда
[адрес1[,адрес2]] { команда1, ... командаN }
```

Адрес определяет строки, к которым применяется команда или команды. Если заданы и первый, и второй адреса, то команда применяется к строкам от первого до второго адреса включительно. Если задан только первый адрес, то команда применяется к строкам, определяемым этим адресом. Если адреса не заданы, то команда применяется ко всем строкам файла.

Адрес может задаваться:

– номером строки;

– регулярным выражением, заключенным в символы «/.../» (подробное описание регулярных выражений приведено в 7.6);

– **c**`regexp`**c** Соотвествует строкам подпадающим под определение регулярного выражения `regexp`. **c** может быть любым символом;

– `<первая>~<шаг>` — соответствует каждой строке `<шаг>` начиная от строки `<первая>`;

– специальным символом «$», адресующим последнюю строку файла.

7.5.28.2. Команды редактирования

– `p` — вывод адресуемых строк на печать;

– `P` — распечатать до первого вхождения новой строки текущей области шаблона;

– = — вывод на печать номеров адресуемых строк;

– { — начало блока команд, должно оканчиваться символом }.

– a — добавление текста после адресуемых строк. Формат команды:

```
[адрес]a\
текст \
. . .
текст
```

– i — добавление текста перед адресуемыми строками. Формат команды:

```
[адрес]i\
текст \
. . .
текст
```

– c — замена блока адресуемых строк заданным текстом. Формат команды:

```
[адрес]c\
текст \
. . .
текст
```

– d — удаление адресуемых строк;

– D — удалить до первого вхождения новой строки в области шаблона;

– s — замена в адресуемых строках заданного шаблона заданным текстом. Формат команды:

```
[адрес]s/<шаблон>/текст/[флаги]
```

<шаблон> задается регулярным выражением (подробное описание регулярных выражений приведено в 7.6).

В тексте может использоваться метасимвол «&» для обозначения заменяемого текста.

Возможные флаги команды s:

1) p — вывод на печать строк, в которых была произведена замена;

2) число — замена задаваемого числом вхождения шаблона (по умолчанию заменяется только первое вхождение);

3) g — замена всех вхождений шаблона в строку;

4) w <файл> — вывод измененных строк в файл;

– y/<источник>/<цель>/ — транслитерация символов в области шаблона, которые присутствуют в источнике с соответствующими символами в цели;

– h, H — скопировать/добавить область шаблон к удерживаемой области;

– g, G — скопировать удерживаемую область к области шаблона;

– x — поменять местами удерживаемую область и область шаблона;

– l — огласить текущую строку в «визуально однозначной» форме;

– n, N — прочитать/добавить следующую строку ввода в область шаблона;

– b <метка> — переход к метке; если метка опущена, переход к концу сценария;

– t <метка> — если конструкция s/.../.../ выполнила успешную замену, так как была прочитана последняя строка ввода и последняя команда t, то выполнить переход к метке; если метка опущена, то перейти к концу сценария;

– w <файл> — вывод адресуемых строк в файл;

– r <файл> — чтение текста из файла и вставка его после каждой адресуемой строки;

– q — завершение работы при достижении адресуемой строки.

7.5.28.3. Примеры

Пространство между строками.

Двойное пространство между строками:

```
$ sed G
```

Двойное пространство между строками исключая пустые строки (на выходе содержатся не больше одной пустой строки между двумя строками с текстом):

```
$ sed '/^$/d;G'
```

Тройное пространство между строками:

```
$ sed 'G;G'
```

Удалить каждую вторую строку:

```
$ sed 'n;d'
```

Вставить пустую строку перед каждой строкой соответствующей регулярному выражению regex:

```
$ sed '/regex/{x;p;x;}'
```

Вставить пустую строку после каждой строки соответствующей регулярному выражению regex:

```
$ sed '/regex/G'
```

Вставить пустую строку перед и после каждой строки соответствующей регулярному выражению regex:

```
$ sed '/regex/{x;p;x;G;}'
```

Нумерация.

Нумерация каждой стоки в файле `filename`, используя отступ вместо пустой строки:

```
$ sed = filename | sed 'N;s/\n/\t/'
```

Нумерация каждой строки в файле `filename` (номер слева, выровненный по правому краю):

```
$ sed = filename | sed 'N; s/^/ /; s/ *\(.\{6,\}\)\n/\1 /'
```

Нумерация каждой строки в файле `filename`, с выводом номера только для не пустых строк:

```
$ sed '/./=' filename | sed '/./N; s/\n/ /'
```

Подсчет строк:

```
$ sed -n '$='
```

Преобразование и замена текста.

Удалить все пробелы и символы табуляции в начале каждой строки файла:

```
$ sed 's/^[ \t]*//'
```

Удалить все пробелы и символы табуляции в конце каждой строки файла:

```
$ sed 's/[ \t]*$//'
```

Удалить все пробелы и символы табуляции в начале и конце каждой строки файла:

```
$ sed 's/^[ \t]*//;s/[ \t]*$//'
```

Вставить 5 пробелов в начале каждой строки (создать смещение страницы):

```
$ sed 's/^/ /'
```

Расположить весь текст по правому краю столбца шириной в 79 символов:

```
$ sed -e :a -e 's/^.\{1,78\}$/ &/;ta'
```

Центрировать весь текст посередине столбца шириной 79 символов. В версии 1 пробелы добавляются в начало и конец строки. В версии 2 пробелы добавляются только в начало строки:

Версия 1:

```
$ sed -e :a -e 's/^.\{1,77\}$/ & /;ta'
```

Версия 2:

```
$ sed -e :a -e 's/^.\{1,77\}$/ &/;ta' -e 's/\( *\)\1/\1/'
```

Поиск и замена `foo` на `bar` в каждой строке.

Замена только первого совпадения в строке:

```
$ sed 's/foo/bar/'
```

Замена первых четырёх совпадений в строке:

```
$ sed 's/foo/bar/4'
```

Замена всех совпадений в строке:

```
$ sed 's/foo/bar/g'
```

Замена предпоследнего совпадения:

```
$ sed 's/\(.*\)foo\(.*foo\)/\1bar\2/'
```

Замена только последнего совпадения:

```
$ sed 's/\(.*\)foo/\1bar/'
```

Замена `foo` на `bar` только для строк содержащих `baz`:

```
$ sed '/baz/s/foo/bar/g'
```

Замена `foo` на `bar` исключая строки содержащие `baz`:

```
$ sed '/baz/!s/foo/bar/g'
```

Замена `scarlet` или `ruby`, или `puce` на `red`:

```
$ sed 's/scarlet\|ruby\|puce/red/g'
```

Перевернуть каждую строку в файле задом наперед:

```
$ sed '/\n/!G;s/\(.\)\(.*\n\)/&\2\1/;//D;s/.//'
```

Соединить каждую пару строк бок о бок:

```
$ sed '$!N;s/\n/ /'
```

Если строка заканчивается обратной косой чертой «\», то присоединить следующую строку:

```
$ sed -e :a -e '/\\$/N; s/\\\n//; ta'
```

Если строка начинается с знака «=», то присоединить её к предыдущей строке и заменить «=» пробелом:

```
$ sed -e :a -e '$!N;s/\n=/ /;ta' -e 'P;D'
```

Добавить запятые к строке из чисел, заменяя «1234567» на «1,234,567»:

```
$ sed ':a;s/\B[0-9]\{3\}\>/,&/;ta'
```

Добавить запятые к числу с десятичной частью и знаком минуса:

```
$ sed -r ':a;s/(^|[^0-9.])([0-9]+)([0-9]{3})/\1\2,\3/g;ta'
```

Добавить пустую строку через каждые 5 строк (после строк 5, 10, 15, 20, и т. д.):

```
$ sed '0~5G'
```

Выборочная печать некоторых строк.

Печатать первые 10 линий файла (эмуляция `head`):

```
$ sed 10q
```

Печатать первую строку файла (эмуляция `head -1`):

```
$ sed q
```

Печатать последние 10 строк файла (эмуляция `tail`):

```
$ sed -e :a -e '$q;N;11,$D;ba'
```

Печатать последние 2 строки файла (эмуляция `tail -2`):

```
$ sed '$!N;$!D'
```

Печатать последнюю строку файла (эмуляция `tail -1`):

```
$ sed '$!d'
```

или

```
$ sed -n '$p'
```

Печатать предпоследнюю строку в файле:

```
$ sed -e '$!{h;d;}' -e x //для однострочного файла печатать пустую строку
$ sed -e '1{$q;}' -e '$!{h;d;}' -e x //для однострочного файла печатать эту строку
$ sed -e '1{$d;}' -e '$!{h;d;}' -e x //для однострочного файла ничего не печатать
```

Печатать только те строки, которые совпадают с регулярным выражением (эмуляция `grep`):

```
$ sed -n '/regexp/p'
```

или

```
$ sed '/regexp/!d'
```

Печатать только те строки, которые не совпадают с регулярным выражением (эмуляция `grep -v`):

```
$ sed -n '/regexp/!p'
```

или

```
$ sed '/regexp/d'
```

Печатать строку непосредственно перед регулярным выражением, но не печатать строку, содержащую регулярное выражение:

```
$ sed -n '/regexp/{g;1!p;};h'
```

Печатать строку непосредственно после регулярного выражения, но не печатать строку, содержащую регулярное выражение:

```
$ sed -n '/regexp/{n;p;}'
```

Печатать по одной строке перед и после регулярного выражения, с указанием номера строки, совпадающей с регулярным выражением (аналог `grep -A1 -B1`):

```
$ sed -n -e '/regexp/{=;x;1!p;g;$!N;p;D;}' -e h
```

Печать строк, совпадающих с регулярными выражениями `AAA`, `BBB` и `CCC` одновременно (в любой последовательности):

```
$ sed '/AAA/!d; /BBB/!d; /CCC/!d'
```

Печать строк, совпадающих с регулярными выражениями `AAA`, `BBB` и `CCC` одновременно (в конкретной последовательности):

```
$ sed '/AAA.*BBB.*CCC/!d'
```

Печать строк, совпадающих с любым регулярным выражением `AAA` или `BBB`, или `CCC` (эмуляция `egrep`):

```
$ sed '/AAA\|BBB\|CCC/!d'
```

Печатать абзац, если он содержит `AAA` (пустая строка разделяет абзацы):

```
$ sed -e '/./{H;$!d;}' -e 'x;/AAA/!d;'
```

Печатать абзац если он содержит `AAA`, `BBB` и `CCC` (в любой последовательности):

```
$ sed -e '/./{H;$!d;}' -e 'x;/AAA/!d;/BBB/!d;/CCC/!d'
```

Печатать строки длиной равной или большей 65 символов:

```
$ sed -n '/^.\{65\}/p'
```

Печатать строки длиной меньше или равной 65 символов:

```
$ sed -n '/^.\{65\}/!p'
```

или

```
$ sed '/^.\{65\}/d'
```

Печатать часть файла, начиная от совпадения с регулярным выражением и до конца файла:

```
$ sed -n '/regexp/,$p'
```

Печатать часть файла, основываясь на номерах строк (строки 8-12, включительно):

```
$ sed -n '8,12p'
```

или

```
$ sed '8,12!d'
```

Печатать строку под номером 52:

```
$ sed -n '52p'
```

или

```
$ sed '52!d'
```

быстрая версия:

```
$ sed '52q;d'
```

Печатать часть файла между двумя регулярными выражениями (включительно):

```
$ sed -n '/Iowa/,/Montana/p'
```

Избирательное удаление определенных строк.

Печатать все строки, исключая часть между двумя регулярными выражениями:

```
$ sed '/Iowa/,/Montana/d'
```

Удалить дубликаты последовательных строк в файле (эмуляция `uniq`). Первая строка в наборе дубликатах строк удерживается от удаления:

```
$ sed '$!N; /^\(.*\)\n\1$/!P; D'
```

Удалить дубликаты непоследовательных строк в файле:

```
$ sed -n 'G; s/\n/&&/; /^\([ -~]*\n\).*\n\1/d; s/\n//; h; P'
```

Печатать только дубликаты строк (эмуляция `uniq -d`):

```
$ sed '$!N; s/^\(.*\)\n\1$/\1/; t; D'
```

Удалить первые 10 строк в файле:

```
$ sed '1,10d'
```

Удалить последнюю строку в файле:

```
$ sed '$d'
```

Удалить 2 последние строки файла:

```
$ sed 'N;$!P;$!D;$d'
```

Удалить последние 10 строк файла:

```
$ sed -e :a -e '$d;N;2,10ba' -e 'P;D'
```

или

```
$ sed -n -e :a -e '1,10!{P;N;D;};N;ba'
```

Удалить каждую восьмую строку в файле:

```
$ sed '0~8d'
```

Удалить строки, совпадающие с регулярным выражением:

```
$ sed '/pattern/d'
```

Удалить все пустые строки из файла (эмуляция `grep '.'`):

```
$ sed '/^$/d'
```

или

```
$ sed '/./!d'
```

Удалить все последовательности пустых строк из файла, исключая первую. Также удалить все пустые строки в начале и в конце файла (эмуляция `cat -s`):

```
$ sed '/./,/^$/!d' //допускается 0 пустых строк в начале и 1 в конце файла
$ sed '/^$/N;/\n$/D' //допускается 1 пустая строка в начале и 0 в конце файла
```

Оставить последовательность пустых строк не более двух одновременно:

```
$ sed '/^$/N;/\n$/N;//D'
```

Удалить все пустые строки в начале файла:

```
$ sed '/./,$!d'
```

Удалить все пустые строки в конце файла:

```
$ sed -e :a -e '/^\n*$/{$d;N;ba' -e '}'
```

Удалить последнюю непустую строку в каждом абзаце:

```
$ sed -n '/^$/{p;h;};/./{x;/./p;}'
```

7.5.29. `awk`

Утилита `awk` ищет в файле строки, удовлетворяющие шаблонам, заданным в `<сценарий awk>`, и выполняет над ними действия, заданные в `<сценарий awk>`.

`awk` разбивает файл на записи (по умолчанию — строки), каждую запись — на поля, используя разделитель (по умолчанию — по пробелу, несколько идущих подряд разделителей объединяются). Каждая запись обрабатывается сценарием. `awk` адресует поля по специальному символу «$»: `$0` — вся строка, `$1` — первое поле, `$2` — второе поле и т. д.

Если не указано ни одного файла или вместо имени файла указан символ дефиса — «-», используется стандартный ввод. Результат работы `awk` направляется в выходной поток.

Формат команды:

```
awk [<параметры>] [<сценарий awk>] [<файл...>]
```

Параметры:

`-F <символ>` Указывает символ, используемый в обрабатываемом тексте как разделитель полей. По умолчанию — пробел.

`-f <файл>` Указывает имя файла, из которого читается `<сценарий awk>`. Если Этот параметр задана, `<сценарий awk>` в командной строке не вводится.

`<сценарий awk>` представляет собой программу на специальном языке `awk`, описывающую шаблоны, по которым отбираются строки и действия над ними. Скрипт представляет собой последовательность элементов, каждый из которых имеет вид:

```
[<шаблон>] {<действие>}
```

Семантика элемента такова, что если находится строка, соответствующая данному шаблону, то над ней выполняется данное действие. Если шаблон пропущен, то действие выполняется над всеми строками файла. Описание шаблонов основывается на базовых регулярных выражениях (РВ), описание действий — на синтаксисе языка программирования С. При работе `awk` читает файл последовательно, строку за строкой и над строками, удовлетворяющими заданным в скрипте шаблонам, выполняет заданные действия.

7.5.29.1. Описания шаблонов

Шаблон представляет собой РВ, заключенное в символы «/.../» (подробное описание регулярных выражений приведено в разделе 7.6). В языке `awk` синтаксис базовых РВ расширен следующими дополнениями:

– `()` — скобки для группирования РВ;

– `|` — логическое «или»;

– `+` — плюс, стоящий за РВ, означает любую последовательность вхождений этого РВ, начиная с первого;

– `?` — знак вопроса, стоящий за РВ, означает 0 или 1 вхождений этого РВ.

В шаблоне также допускаются выражения отношения, которые имеют вид:

```
<выражение> <операция принадлежности> <выражение>
```

или

```
<выражение> <операция отношения> <выражение>
```

Операции принадлежности бывают: «~» (принадлежит) и «!~» (не принадлежит). Операции отношения: «==», «!=», «>», «>=», «<», «<=» — в их обычном смысле. В левой части таких выражений в обоих случаях обычно применяется имя поля строки, в правой, в первом случае — шаблон, во втором — любое выражение.

Допускается логическая комбинация шаблонов с использованием операций «`&&`», «`||`», «`!`».

Комбинация вида:

<шаблон1>, <шаблон2>

означает применение задаваемых с данными шаблонами действий к строке, удовлетворяющей <шаблон1>, и далее — ко всем следующим за ней строкам, вплоть до появления строки, удовлетворяющей <шаблон2>, включительно.

В языке awk предусмотрены два специальных шаблона — BEGIN и END. Первый описывает действия, выполняемые перед началом чтения файла, второй — действия, выполняемые после окончания чтения.

7.5.29.2. Действия

Язык описания действий awk почти идентичен языку программирования C.

Операции:

```
+     -     *     /     %
++    --    в постфиксной и префиксной формах
=     +=    -=    *=    /=    % =
<     <=    >     >=    ==    !=
!     &&    ||
```

Оператор, последняя операция в котором является операцией присваивания, является оператором присваивания.

Операторы, управляющие потоком вычисления:

```
if (условие) оператор
[else оператор]

while (условие) оператор

for (выражение; условие; выражение) оператор
     в отличие от языка C, в выражениях цикла for не
допускается перечисление через запятую

break

continue
```

```
next    немедленный переход к следующей строке файла

exit    выход из программы
```

Операторы вывода:

– `print <список выражений>` — выводит значения выражений, перечисленных в списке (через пробел);

– `printf (...)` — аналог одноименной функции языка С.

Оператор завершается символом «`;`» или переводом строки. Если оператор будет продолжен на следующей строке, первая строка должна завершаться символом «\».

Любая последовательность операторов, заключенная в фигурные скобки `{...}` является составным оператором.

Комментарий имеет тот же вид, что и в языке С: `/*...*/`, в отличие от С, комментарии можно вставлять только между операторами, но не в середину оператора.

Функции.

`asort(s[,d])` Сортирует массив `s` и возвращает количество элементов в массиве. Если задан параметр `d`, то массив `s` сначала копируется в `d`, после чего `d` сортируется, оставляя массив `s` нетронутым.

`asorti(s[,d])` Аналогична `asort` за исключением, что сортировка массива происходит по ключу, а не по значению.

`length(arg)` Возвращает длину `arg`. Если `arg` не указан, то выдает длину текущей строки.

`exp(), log(), sqrt()` Математические функции: экспонента, логарифм, квадратный корень.

`int()` Возвращает целую часть числа.

`substr(s,m,n)` Возвращает подстроку строки `s`, начиная с позиции `m`, всего `n` символов. Если `n` не задано — до конца строки.

`index(s,t)` Возвращает начальную позицию подстроки `t` в строке `s` или 0, если `t` в `s` не содержится.

`sprintf(fmt,exp1,exp2,...)` Форматированная печать в строку, идентично `printf()`.

`split(s,array,sep)` Помещает поля строки `s` в массив `array` и возвращает число заполненных элементов массива. Если указан `sep`, то при анализе строки он понимается как разделитель.

`strtonum(s)` Преобразует строку в число. Если строка начинается с «0», подразумевается, что число задано в восьмеричном формате. Если строка начинается с «0x» или «0X», подразумевается, что число записано в шестнадцетиричном формате.

`systime()` Возвращает время в формате UNIX.

`toupper(s)` Переключает строку в верхний регистр.

`tolower(s)` Переключает строку в нижний регистр.

Язык программирования `awk` допускает использование:

– полей;

– стандартных переменных;

– пользовательских переменных;

– массивов (в том числе ассоциативных).

Ссылки на поля обрабатываемой строки возможны по именам: `$1`, `$2`, `$3` и т. п. `$0` — ссылка на всю строку.

В языке `awk` предопределены стандартные переменные, перечисленные в таблице 10.

Таблица 10 – Перечень переменных языка `awk`

Переменная	Описание
`FILENAME`	Имя текущего обрабатываемого файла
`FS`	Разделитель полей во входной строке (по умолчанию — пробел)
`NF`	Число полей во входной строке
`NR`	Номер текущей входной строки (количество просмотренных записей)
`OFS`	Разделитель полей в выводе (по умолчанию — пробел)
`ORS`	Разделитель записей в выводе
`ARGC`	Количество аргументов командной строки
`ARGV`	Массив значений аргументов командной строки
`RS`	Разделитель записей (по умолчанию — символ новой строки)
`ORS`	Разделитель записей в выводе (по умолчанию — символ новой строки)

Пользовательские переменные не требуют объявления, они автоматически объявляются при их появлении в программе. Переменные могут интерпретировать-

ся как числовые или строковые, интерпретация выполняется в зависимости от контекста использования переменной.

Массивы также не объявляются, а принимают значения из контекста. Массивы в скрипте `awk` являются динамическими, то есть, новые элементы добавляются в массив по мере необходимости. Индексом в массиве может быть как числовое, так и строковое значение.

Примеры.

Возьмём входной файл `f-awk` (фамилия, инициалы, год приёма на работу, возраст):

```
Иванов И.И.   1980  50
Петров А.В.   1979  40
Сидоров С.К.  1979  40
Хведоров И.Х. 1970  60
```

Вывести весь текст:

```
$ awk '{print}' f-awk
```

Вывести все строки, в которых содержится подстрока «до»:

```
$ awk '/до/ {print}' f-awk
Сидоров С.К.  1979  40
Хведоров И.Х. 1970  60
```

Следующие команды идентичны предыдущей:

```
$ cat f-awk | awk '/до/ {print}'
$ awk '/до/ {print}' < f-awk
```

Вывести только второе поле (инициалы):

```
$ awk  '/до/ {print $2}'  f-awk
С.К.
И.Х.
```

Если изменить используемый разделитель на точку, вторым полем будет вторая буква инициалов:

```
$ awk -F. '/до/ {print $2}'  f-awk
К
Х
```

То есть `awk` разбивает строки на поля следующим образом:

```
Сидоров С.К.  1979  40
---------^-^----------
1-е поле  2 3-е поле
```

Операции над числами (сумма возраста и года):

```
$ awk ’/до/ {print $3+$4}’  f-awk
2019
2030
```

Год и возраст по отдельности:

```
$ awk ’/до/ {print $3, $4}’  f-awk
1979 40
1970 60
```

Средний возраст (возраст каждого суммируется, в конце нужно разделить на количество записей):

```
$ awk ’{s+=$4}END{print s/NR}’  f-awk
47.5
```

Вывести стаж каждого:

```
$ awk ’{print ($1, 2000 - $3)}’ f-awk
Иванов 20
Петров 21
Сидоров 21
Хведоров 30
```

Выборка по регулярному выражению (только те строки, в которых третье поле — год — семидесятые):

```
$ awk ’$3~/7[0-9]$/ {print}’ f-awk
Петров А.В.   1979  40
Сидоров С.К.  1979  40
Хведоров И.Х. 1970  60
```

Выборка по строгому равенству:

```
$ awk ’$1==”Иванов” {print} ’ f-awk
Иванов И.И.   1980  50
```

Математическая выборка:

```
awk ’$3 != $4 && $3 > 1970 {print}’ f-awk
Иванов И.И.   1980  50
Петров А.В.   1979  40
Сидоров С.К.  1979  40
```

7.6. Регулярные выражения

Регулярные выражения (далее — РВ) — формальный язык для обработки текстовых строк с использованием метасимволов. РВ представляет собой шаблон, которому строка или часть строки могут соответствовать, тогда говорят о соответствии строки данном РВ, либо не соответствовать, в таком случае говорят, что строка не соответствует данному РВ.

РВ широко используются при обработке текста. Они позволяют осуществлять сложную выборку текста в тех случаях, когда не подходят стандартные функции поиска. В ОС РВ могут использоваться в следующих программах:

– `grep`;

– `sed`;

– `awk`;

– языки программирования (Perl, Python, PHP).

РВ позволяют решать самый широкий круг задач по обработке текста. Некоторые из них:

– поиск в тексте слов и словоформ;

– выборка из XML документа некоторой информации;

– анализ файлов журналов на соответствие некоторым событиям, таким как «ошибка доступа к веб-серверу для IP-адресов из подсети 192.168».

Регулярные выражения в стиле языка Perl поддерживаются большинством программ и именно данный синтаксис будет рассмотрен ниже.

7.6.1. Базовые понятия

Регулярные выражения используются для сжатого описания некоторого множества строк с помощью шаблонов, без необходимости перечисления всех элементов этого множества. При составлении шаблонов используется специальный синтаксис, поддерживающий, обычно, следующие операции:

– перечисления — шаблону будет соответствовать некоторый набор символов или строк;

– квантификаторы — квантификатор определяет, сколько раз может встречаться предшествующее выражение;

– группы — группы позволяют как осуществлять логическую группировку в выражении, так и осуществлять поиск с заменой, запоминая содержимое каждой группы в буфере.

7.6.2. Символы

Описание групп символов приведено в таблице 11.

Таблица 11 – Специальные символы регулярных выражений

Символ	Описание
^	Начало строки
$	Конец строки
.	Любой символ, кроме «\n» — конец строки
|	Оператор альтернативы (или)
()	Группировка
[]	Набор символов или диапазон
\	Экранирование — ставится перед спецсимволом для того, чтобы считать его частью шаблона
\t	Символ табуляции
\n	Новая строка
\r	Перевод каретки
\a	Перевод формата
\v	Вертикальная табуляция
\a	Звонок (символ bell)
\e	Escape
\033	Восьмеричная запись символа
\x1A	Шестнадцатеричная запись символа
\c[	Символ Control
\l	Нижний регистр следующего символа
\u	Верхний регистр следующего символа
\L	Все символы в нижнем регистре до «\E»
\U	В верхнем регистре до «\E»
\E	Ограничитель смены регистра
\Q	Отмена действия как метасимвола
\w	Алфавитно-цифровой символ и символ «_»

Символ	Описание
\W	Любой символ, кроме алфавитно-цифрового и символа «_»
\s	Один пробел
\S	Любой один символ кроме пробела
\d	Одна цифра
\D	Любой один символ кроме цифры
\b	Граница слова
\B	Не граница слова
\A	Начало строки
\Z	Конец строки

В квадратных скобках допускается перечислять любые наборы символов, однако сами квадратные скобки должны быть экранированы. Кроме того, можно использовать любые другие метасимволы, например, \s или \w внутри квадратных скобок и диапазоны символов, например: [a-z] — любой символ от a до z.

Каждый из представленных символов — это всего лишь один символ, например, набор [a-z] означает один символ от a до z и не будет соответствовать слову из двух и более букв. Для соответствия целым словам необходимо использовать квантификаторы, описанные в следующем разделе.

Некоторые символы являются мнимыми, такие как границы слов и строк. Они просто дают подсказку регулярному выражению, в каком месте строки или слова должен находиться шаблон, например, выражению «cat$» будет соответствовать строка «cat» или «scat», но не будет соответствовать строка «catalog».

7.6.3. Квантификаторы

Список квантификаторов представлен в таблице 12.

Таблица 12 – Квантификаторы регулярных выражений

Квантификатор	Описание
*	повторяется 0 или большее число раз
+	повторяется 1 или большее число раз
?	1 или 0 раз
{n}	точно n раз
{n,}	не менее n раз
{,n}	не более n раз

Квантификатор	Описание
`{n,m}`	от `n` до `m` раз
Добавить `?`, чтобы сделать нежадным	
Добавить `+`, чтобы сделать сверхжадным	

П р и м е ч а н и е. `n` и `m` не могут быть больше 65536.

По умолчанию квантификаторы являются «жадными» — они захватывают максимально возможную последовательность символов, не учитывая результат действия следующих метасимволов. Иногда это мешает поиску информации по контрольным точкам (тегам), поэтому для уменьшения «жадности» квантификатора после него можно поставить символ «?». В результате квантификатор будет не только сравнивать строку со своим собственным набором символов, но и с последующими метасимволами.

Например, выражение «`<.*>`» должно находить все символы внутри угловых скобок и в строке «`thit is a <tag>`» будет найдено слово «`tag`». Однако, такое возражение некорректно отработает на строке «`this is a first <tag>, this is a <second>`» и результатом поиска будет «`tag>, this is a <second`». Если же использовать нежадный квантификатор: «`<.*?>`», то в той же строке будут найдены два слова: «`tag`» и «`second`».

Символ «+» после квантификатора имеет обратный эффект: квантификатор станет сверхжадным, то есть он будет захватывать максимально возможную подходящую под шаблон последовательность символов.

7.7. Утилиты работы с архивами

Основное назначение архиваторов — создание резервных копий и экономия места на диске. Важную роль архиваторы играют при сжатии файлов журналов, которые обычно занимают много места на диске и эффективно сжимаются архиваторами.

Операционная система «ОСь» поддерживает следующие виды архивов:

– `tar` — формат без сжатия;

– `gz` — основной формат сжатия, простой и быстрый, идеально подходит для сжатия файлов журналов;

– `bz2` — более продвинутый алгоритм сжатия, но более медленный, чем `gz`;

– `zip` — совместим с другими операционными системами, по скорости и степени сжатия аналогичен `gz`;

– lzma и xz — самые эффективные и медленные алгоритмы сжатия.

Архиваторы gz, bz2, lzma и xz не предназначены для архивирования каталогов и выполняют сжатие только одного файла. Для того, чтобы упаковать каталог с вложенными в него файлами и подкаталогами, его нужно сначала упаковать в формат tar, после чего сжать одним из доступных архиваторов. Только формат zip поддерживает архивы с файлами и каталогами.

Для удобства не используют утилиты gz, bz2, lzma и xz напрямую, а используют их через программу tar.

Если файл имеет расширение *.tar, это означает, что он упакован только в формате tar. Если расширение файла — *.gz, *.gz, *.bz2, *.xz или *.lzma, это означает, что архив содержит один единственный файл, обычно так упаковывают файлы журналов, которые можно читать напрямую из упакованного вида. Формат *.tar.gz и аналогичные обычно используются для хранения архивов со структурой каталогов — наиболее используемый формат хранения архивов для резервных копий.

7.7.1. tar

Сжатие и распаковка файлов производится утилитой tar — наиболее функциональной из всех утилит сжатия данных. Она позволяет быстро создавать или распаковывать архивы вида *.tar.gz.

Формат команды:

tar [параметры] <список файлов и каталогов>

Основные параметры (одновременно может использоваться только один из этих параметров).

-A, --catenate, --concatenate Добавляет файлы в архив.

-c, --create Создает новый архив.

-d, --diff, --compare Находит различия между объектами архива и их ”исходниками” в файловой системе.

--delete Удаляет файлы из архива.

-r, --append Добавляет файлы в конец архива.

-t, --list Выводит содержание архива.

-u, --update Добавляет только те файлы, которые ранее не были включены в архив.

-x, --extract, --get Извлекает файлы из архива.

Дополнительные параметры.

`--atime-preserve` Не меняет время доступа скопированных файлов.

`-b, --block-size N` Устанавливает размер блока в (N x 512) байт (по умолчанию N=20).

`-B, --read-full-blocks` Считывает фактические блоки.

`-C, --directory <каталог>` Меняет каталог на указанный.

`--checkpoint` Выводит имена каталогов при чтении архива.

`-f, --file [<имя хоста>:]<файл>` Файл архива, с которым нужно работать (возможно также удалённая сетевая работа с архивом на узле `<имя хоста>`).

`--force-local` Предполагает, что архивный файл расположен в локальной системе, даже если за параметром `-f` имеется двоеточие.

`-g, --listed-incremental=<файл>` Создаёт инкрементный архив в формате GNU (старый).

`-G, --incremental=<файл>` Создаёт инкрементный архив в формате GNU (новый).

`-h, --dereference` Не дублирует сами символические ссылки, лишь дублирует файлы, на которые они указывают.

`-i, --ignore-zeros` Игнорирует в архиве блоки нулей (обозначающие обычно EOF — End Of File).

`-j, --bzip` Обрабатывает архив с помощью архиватора `bzip2`.

`-J, --xz` Обрабатывает архив с помощью архиватора `xz`.

`--ignore-failed-read` Игнорирует не читаемые файлы (т .е. не завершает работу с кодом, отличным от нуля).

`-k, --keep-old-files` Сохраняет существующие файлы и не перекрывает их при извлечении объектов архива.

`-K, --starting-file <файл>` Начинает обработку архива с файла `<файл>`.

`-l, --one-file-system` При создании архива остается в локальной файловой системе.

`-m, --modification-time` При извлечении объектов архива игнорирует их время модификации.

`-M, --multi-volume` Работает (создает, отображает или извлекает) с многотомным архивом.

`-N, --after-date <дата>, --newer <дата>` Сохраняет только более новые файлы относительно даты `<дата>`.

`-O, --to-stdout` Направляет извлекаемые из архива файлы на стандартный вывод.

`-p, --same-permissions, --preserve-permissions` Извлекает информацию о правах доступа.

`-P, --absolute-paths` Сохраняет абсолютный путь имён файлов.

`--preserve` Соответствует одновременному заданию параметров `-p -s`.

`-R, --record-number` Отображает номер записи архивной информации с каждым сообщением.

`--remove-files` Удаляет исходные (оригинальные) файлы после добавления их в архив.

`-s, --same-order, --preserve-order` Сортирует список имен файлов, которые извлекаются из архива.

`--same-owner` При извлечении файлов из архива сохраняет поле владельца файла.

`-S, --sparse` Оптимально поддерживает «распределённые (разбросанные)» файлы.

`-T, --files-from=<файл>` Для извлечения или создания получает имена из файла `<файл>`.

`--null` Считывает заканчивающиеся нулем имена файлов. Запрещает параметр `-C`.

`--totals` Выводит общее число байт, записанное с помощью параметра `--create`.

`-v, --verbose` предоставляет список обрабатываемых файлов.

`-w, --interactive, --confirmation` Запрашивает подтверждение для каждого действия.

`-W, --verify` Пытается выполнять проверку архива после записи в него.

`--exclude <файл>` Исключает из обработки файл `<файл>`.

`-X, --exclude-from <файл>` Исключает из обработки файлы, перечисленные в файле `<файл>`.

`-z, --gzip, --ungzip` Обрабатывает архив с помощью архиватора `gzip`.

`--lzma` Обрабатывает архив с помощью архиватора `lzma`.

`--use-compress-program <программа>` Обрабатывает архив с помощью программы `<программа>` (которая должна принимать параметр `-d`).

`--xattrs` Сохранять расширенные атрибуты файловой системы.

Пример создания архива `gzip`:

```
tar -cvzf arch.tar.gz file1 dir2
```

Пример создания архива `bz2`:

```
tar -cvjf arch.tar.bz2 file1 dir2
```

Пример создания архива `xz`:

```
tar -cvJf arch.tar.xz file1 dir2
```

Пример распаковки архива (любой формат):

```
tar -xvf arch.tar.gz
```

7.7.2. `cpio`

Программа `cpio` предназначена для копирования файлов в архив и из архива `cpio`.

7.7.2.1. Режим копирования из архива

`cpio -i` (copy in) — выбирает файлы из стандартного входного потока, предположительно сформированного предыдущей командой `cpio -o`. Выбираются только файлы с именами, соответствующими шаблону Bash (см. раздел 4.2.8.6). Извлечённые файлы при необходимости создаются и копируются в текущее дерево каталогов. Права доступа к файлам будут такими же, как и в момент выполнения соответствующей команды `cpio -o`. Пользователь и группа-владелец устанавливаются на основе текущего пользователя, если только это не системный администратор. В этом случае владельцы будут такие же, как были при выполнении соответствующей команды `cpio -o`. Если команда `cpio -i` пытается создать файл, который уже существует, причем с той же датой изменения или более новый, команда `cpio` выдаст предупреждающее сообщение и не заменит этот файл. Для безусловной перезаписи существующих файлов можно задать параметр `-u`.

7.7.2.2. Режим копирования в архив

`cpio -o` (copy out) — читает список имен файлов из входного потока и копирует эти файлы в стандартный выходной поток вместе с полным путём и информа-

цией о состоянии. Результат по умолчанию дополняется до границы 8192 байт или до указанного пользователем (с помощью параметров `-B` или `-C`) размера блока, или, при необходимости, до специфического размера блока устройства (как в случае использования ленты СТС).

7.7.2.3. Режим передачи

`cpio -p` (pass) — читает список имён файлов, которые при необходимости, создаются и копируются в указанное целевое дерево каталогов.

7.7.3. `ar`

Программа `ar` создает и модифицирует архивы, а также извлекает компоненты из архива. Архив — набор объектов (файлов), которые называются компонентами.

Содержимое оригинального файла, права доступа, временные метки, владелец и группа сохраняются в архиве и могут быть восстановлены при извлечении.

`ar` является бинарной утилитой, архивы этого вида очень часто используют как библиотеки, содержащие часто употребляющиеся подпрограммы.

Если задать модификатор `s`, то `ar` будет создавать индекс для символов определенных в объектных модулях, содержащихся в архиве. Однажды созданный индекс освежается в архиве, когда `ar` изменяет содержимое этого архива (сохраняется для операции обновления `q`). Архив с индексом быстрее присоединяется к библиотеке и позволяет подпрограммам в библиотеке вызывать друг друга вне зависимости от их размещения в архиве.

`nm -s` или `nm --print-armap` может использоваться для получения таблицы индекса. Если архив не содержит эту таблицу, то можно использовать `ranlib` (другую форму `ar`) для получения данной таблицы.

Формат команды:

```
ar [-]p[<модификатор> [<имя компонента архива>] [<счётчик>]] <архив> [<файл...>]
```

Параметры выполнения (нужно указать только один из них).

`d` Удаляет модули из архива.

`m` Перемещение компонентов в архив.

`p` Выводит заданные компоненты архива на стандартный вывод.

`q` Быстрое добавление, добавляет файлы в конец архива без проверки на замещение.

`r` Вставляет файлы компонент в архив (с замещением).

`t` Показывает таблицу, в которой хранится содержание архива.

`x` Извлекает компоненты из архива.

Следующие модификаторы могут быть использованы непосредственно после параметра операции, для определения поведения выполняемых операций:

`a` Добавить новые файлы после одного из существующих в архиве компонентов.

`b` Добавить новый файл перед одним из существующих в архиве компонентов.

`c` Создать архив.

`f` Урезает имена в архиве.

`i` Вставить новые файлы перед существующим в архиве компонентом.

`N` Извлекает или удаляет указанное имя из архива заданное счётчиком количество раз.

`o` Восстанавливает оригинальные даты компонентов архива при их извлечении.

`P` Использует полное имя файла при сравнении имён в архиве.

`s` Записывает индекс объектного файла в архив или, если он существует, обновляет его, даже если нет других изменений в архиве. Запуск `ar s` эквивалентен запуску `ranlib`.

`S` Не генерировать таблицу символов архива.

`u` Вставить только те объекты, которые отличаются от уже имеющихся в архиве.

`v` Включает режим выдачи подробностей выполняемых операций.

`V` Показывает версию `ar`.

7.7.4. Чтение архивных файлов без распаковки

Существует особый тип архивов, в которых содержатся только текстовые данные. Обычно подобные архивы создаются при ротации журналов, где файлы журналов сжимаются для экономии места на диске. Подобные файлы невозможно прочитать утилитами `cat` или `less`.

Для чтения текстовых архивов в ОС присутствует набор программ, дублирующий функциональность уже имеющихся стандартных команд (`cat`, `less`, `grep`, `more`, `diff`).

Для чтения архива формата `gzip` используйте следующие команды:

- `zcat`
- `zcmp`
- `zdiff`
- `zegrep`
- `zfgrep`
- `zgrep`
- `zless`
- `zmore`

Для чтения архива формата `bzip2` используйте следующие команды:

- `bzcat`
- `bzcmp`
- `bzdiff`
- `bzgrep`
- `bzless`
- `bzmore`

Для чтения архива формата `xz` используйте следующие команды:

- `xzcat`
- `xzcmp`
- `xzdiff`
- `xzegrep`
- `xzfgrep`
- `xzgrep`
- `xzless`
- `xzmore`

Для чтения архива формата `lzma` используйте следующие команды:

- `lzcat`
- `lzcmp`
- `lzdiff`
- `lzegrep`
- `lzfgrep`
- `lzgrep`
- `lzless`
- `lzmore`

Использование подобных команд позволяет работать с архивированными журналами без необходимости их предварительной распаковки. Дальнейшую обработку и чтение файла удобно производить в конвейере, например:

```
$ bzcat file.bz2 | grep foo | sort -u | less
```

7.8. Менеджер логических томов

7.8.1. Архитектура

МЛТ — это система управления дисковым пространством, абстрагирующаяся от физических устройств. Она позволяет эффективно использовать и легко управлять дисковым пространством. МЛТ обладает хорошей масштабируемостью, уменьшает общую сложность системы. У логических томов, созданных с помощью МЛТ (далее по тексту МЛТ будет обозначатся как LVM от английского «Logical Volume Manager»), можно легко изменить размер, а их названия могут нести большую смысловую нагрузку, в отличие от традиционных `/dev/sda`, `/dev/hda` и т. п.

7.8.1.1. Терминология

Поскольку система управления логическими томами использует собственную модель представления дискового пространства, необходимо определиться с терминами и взаимосвязями понятий.

Рассмотрим схему, демонстрирующую взаимосвязь понятий системы LVM:

```
sda1     sda2     sdb     sdc       <-- PV - Физический том
 |        |        |       |
 |        |        |       |
 +--------+- VG00 -+-------+       <-- VG - Группа томов
              |
 +-------+-------+---------+
 |       |       |         |
root    usr     home      var       <-- LV - Логический том
 |       |       |         |
ext3 reiserfs reiserfs    xfs       <-- Файловые системы
```

7.8.1.2. Обозначения и понятия

– PV, Physical volume, физический том. Обычно это раздел на диске или весь диск. В том числе, устройства программного и аппаратного RAID (которые уже могут включать в себя несколько физических дисков). Физические тома входят в состав группы томов.

– VG, Volume group, группа томов. Это самый верхний уровень абстрактной модели, используемой системой LVM. С одной стороны группа томов состоит из физических томов, с другой — из логических и представляет собой единую административную единицу.

– LV, Logical volume, логический том. Раздел группы томов, эквивалентен разделу диска в не-LVM системе. Представляет собой блочное устройство и, как следствие, может содержать файловую систему.

– PE, Physical extent, физический экстент. Каждый физический том делится на порции данных, называющиеся физическими экстентами. Их размеры те же, что и у логических экстентов.

– LE, Logical extent, логический экстент. Каждый логический том делится на порции данных, называющиеся логическими экстентами. Размер логических экстентов не меняется в пределах группы томов.

Пусть имеется группа томов VG00 с размером физического экстента 4 мегабайта. В эту группу мы добавляем два раздела, `/dev/hda1` и `/dev/hdb1`. Эти разделы становятся физическими томами, например PV1 и PV2 (символьные имена присваивает администратор, так что они могут быть более осмысленными). Физические тома делятся на 4-х мегабайтные порции данных, т.к. это размер физического экстента. Диски имеют разный размер: PV1 получается размером в 99 экстентов, а PV2 — размером в 248 экстентов. Теперь можно приступать к созданию логических томов, размером от 1 до 347 (248+99) экстентов. При создании логического тома, определяется отображение между логическими и физическими экстентами. Например, логический экстент 1 может отображаться в физический экстент 51 тома PV1. В этом случае, данные, записанные в первые 4Мб логического экстента 1, будут в действительности записаны в 51-й экстент тома PV1.

Администратор может выбрать алгоритм отображения логических экстентов в физические. На данный момент доступны два алгоритма:

1) линейное отображение последовательно назначает набор физических экстентов области логического тома, т. е. LE 1–99 отображаются на PV1, а LE 100–347 — на PV2;

2) «расслоенное» (striped) отображение разделяет порции данных логических экстентов на определенное количество физических томов. То есть:

1-я порция данных LE[1] -> PV1[1],

2-я порция данных LE[1] -> PV2[1],

3-я порция данных LE[1] -> PV3[1],

4-я порция данных LE[1] -> PV1[2], и т. д.

Похожая схема используется в работе RAID нулевого уровня. В некоторых ситуациях этот алгоритм отображения позволяет увеличить производительность логического тома. Однако он имеет значительное ограничение: логический том с данным отображением не может быть расширен за пределы физических томов, на которых он изначально и создавался.

Великолепная возможность, предоставляемая системой LVM — это «снапшоты». Они позволяют администратору создавать новые блочные устройства с точной копией логического тома, «замороженного» в какой-то момент времени. Обычно это используется в пакетных режимах. Например, при создании резервной копии системы. Однако при этом вам не будет нужно останавливать работающие задачи, меняющие данные на файловой системе. Когда необходимые процедуры будут выполнены, системный администратор может просто удалить устройство-«снапшот». Ниже мы рассмотрим работу с таким устройством.

7.8.2. Работа с LVM

Рассмотрим задачи, стоящие перед администратором LVM системы.

Помните, что для работы с системой LVM ее нужно инициализировать командами:

```
# vgscan
# vgchange -ay
```

Первая команда сканирует диски на предмет наличия групп томов, вторая активирует все найденные группы томов. Аналогично для завершения всех работ, связанных с LVM, нужно выполнить деактивацию групп:

```
# vgchange -an
```

Первые две строки нужно будет поместить в скрипты автозагрузки (если их там нет), а последнюю можно дописать в скрипт `shutdown`.

7.8.2.1. Инициализация дисков и разделов

Перед использованием диска или раздела в качестве физического тома необходимо его инициализировать:

Для целого диска:

```
# pvcreate /dev/hdb
```

Эта команда создает в начале диска дескриптор группы томов.

Если вы получили ошибку инициализации диска с таблицей разделов — проверьте, что работаете именно с нужным диском, и когда полностью будете уверены в том, что делаете, выполните следующие команды:

```
# dd if=/dev/zero of=/dev/diskname bs=1k count=1
# blockdev --rereadpt /dev/diskname
```

Эти команды уничтожат таблицу разделов на целевом диске.

Для разделов:

Установите программой `fdisk` тип раздела в `0x8e`.

```
# pvcreate /dev/hdb1
```

Команда создаст в начале раздела `/dev/hdb1` дескриптор группы томов.

7.8.2.2. Создание группы томов

Для создания группы томов используется команда `vgcreate`:

```
# vgcreate vg00 /dev/hda1 /dev/hdb1
```

Если вы используете `devfs` важно указывать полное имя в `devfs`, а не ссылку в каталоге `/dev`. Таким образом приведенная команда должна выглядеть в системе с `devfs` так:

```
# vgcreate vg00 /dev/ide/host0/bus0/target0/lun0/part1
/dev/ide/host0/bus0/target1/lun0/part1
```

Кроме того, вы можете задать размер экстента при помощи параметра `-s`, если значение по умолчанию в 4 мегабайта не подходит. Можно также указать ограничения возможного количества физических и логических томов.

7.8.2.3. Активация группы томов

После перезагрузки системы или выполнения команды `vgchange -an`, ваши группы томов и логические тома находятся в неактивном состоянии. Для их активации необходимо выполнить команду

```
# vgchange -a y vg00
```

7.8.2.4. Удаление группы томов

Убедитесь, что группа томов не содержит логических томов. Как это сделать, показано в следующих разделах.

Деактивируйте группу томов:

```
# vgchange -a n vg00
```

Теперь можно удалить группу томов командой:

```
# vgremove vg00
```

7.8.2.5. Добавление физических томов в группу томов

Для добавления предварительно инициализированного физического тома в существующую группу томов используется команда `vgextend`:

```
# vgextend vg00 /dev/hdc1
               ^^^^^^^^^ новый физический том
```

7.8.2.6. Удаление физических томов из группы томов

Убедитесь, что физический том не используется никакими логическими томами. Для этого используйте команду `pvdisplay`:

```
# pvdisplay /dev/hda1

--- Physical volume ---
   PV Name               /dev/hda1
   VG Name               vg00
   PV Size               1.95 GB / NOT usable 4 MB [LVM: 122
KB]
   PV#                   1
   PV Status             available
   Allocatable           yes (but full)
```

```
Cur LV                1
PE Size (KByte)       4096
Total PE              499
Free PE               0
Allocated PE          499
PV UUID               Sd44tK-9IRw-SrMC-MOkn-76iP-iftz-OVSen7
```

Если же физический том используется, вам нужно будет перенести данные на другой физический том. Эта процедура будет описана в следующих разделах.

После этого можно использовать `vgreduce` для удаления физических томов:

```
# vgreduce vg00 /dev/hda1
```

7.8.2.7. Создание логического тома

Для того, чтобы создать логический том `lv00`, размером 1500 мегабайт, выполните команду:

```
# lvcreate -L1500 -n lv00 vg00
```

Без указания суффикса размеру раздела используется множитель «мегабайт» (в системе СИ равный 10^6 байт), что и продемонстрировано в примере выше. Суффиксы в верхнем регистре (KMGTPE) соответствуют единицам в системе СИ (с основанием 10), например, G — гигабайт равен 10^9 байт, а суффиксы в нижнем регистре (kmgtpe) соответствуют единицам в системе IEC (с основанием 2), например g — гигабайт равен 2^{30} байт.

Для создания логического тома размером в 100 логических экстентов с расслоением по двум физическим томам и размером блока данных 4 килобайт:

```
# lvcreate -i2 -I4 -l100 -n lv01 vg00
```

Если вы хотите создать логический том, полностью занимающий группу томов, выполните команду `vgdisplay`, чтобы узнать полный размер группы томов, после чего используйте команду `lvcreate`:

```
# vgdisplay vg00 | grep "Total PE"
Total PE 10230
# lvcreate -l 10230 vg00 -n lv02
```

Эти команды создают логический том `lv02`, полностью заполняющий группу томов. То же самое можно реализовать командой:

```
# lvcreate -l100%FREE vg00 -n lv02
```

7.8.2.8. Удаление логических томов

Логический том должен быть размонтирован перед удалением:

```
# umount /dev/vg00/home
# lvremove /dev/vg00/home
lvremove -- do you really want to remove ”/dev/vg00/home”? [y/n]: y
lvremove -- doing automatic backup of volume group ”vg00”
lvremove -- logical volume ”/dev/vg00/home” successfully removed
```

7.8.2.9. Увеличение логических томов

Для увеличения логического тома вам нужно просто указать команде `lvextend` до какого размера вы хотите увеличить том:

```
# lvextend -L12G /dev/vg00/home
lvextend -- extending logical volume ”/dev/vg00/home” to 12 GB
lvextend -- doing automatic backup of volume group ”vg00”
lvextend -- logical volume ”/dev/vg00/home” successfully extended
```

В результате `/dev/vg00/home` увеличится до 13 гигабайт.

```
# lvextend -L+1G /dev/vg00/home
lvextend -- extending logical volume ”/dev/vg00/home” to 13 GB
lvextend -- doing automatic backup of volume group ”vg00”
lvextend -- logical volume ”/dev/vg00/home” successfully extended
```

Эта команда увеличивает размер логического тома на 1 гигабайт.

После того как вы увеличили логический том, необходимо соответственно увеличить размер файловой системы. Как это сделать, зависит от типа используемой файловой системы.

По умолчанию большинство утилит изменения размера файловой системы увеличивают ее размер до размера соответствующего логического тома. Так что вам не нужно беспокоиться об указании одинаковых размеров для всех команд.

7.8.2.10. Перенос данных с физического тома

Для того, чтобы можно было удалить физический том из группы томов, необходимо освободить все занятые на нем физические экстенты. Это делается путем перераспределения занятых физических экстентов на другие физические тома. Следовательно, в группе томов должно быть достаточно свободных физических экстентов.

7.9. Резервное копирование и восстановление

Создание резервных копий имеет очень большое значение при работе с важной информацией. Информация может быть утеряна или повреждена в силу непредвиденных обстоятельств (случайное удаление файлов, сбой оборудования, сбой программного обеспечения, стихийные бедствия).

Администратор в ОС выполняет полное копирование жесткого диска или частичное копирование только важных файлов.

7.9.1. Резервное копирование жёсткого диска или тома

Команда `dd` позволяет сохранить в файл любое блочное устройство путём полного копирования. Перед резервным копированием или восстановлением с помощью `dd` нужно предварительно отмонтировать этот жёсткий диск или том:

```
# umount /dev/sda
```

Только таким образом можно гарантировать, что в процессе резервирования данные на диске не будут изменены.

Резервное копирование жёсткого диска целиком выполняется следующим образом:

```
# dd if=/dev/sda bs=1024k of=/media/storage/backup
```

Стоит отметить, что резервное копирование жёсткого диска с помощью команды `dd` на этот же диск не допускается. Данная процедура возможна только в том случае, если диск разбит на несколько томов. Например, можно выполнить резервное копирование тома `/dev/sda1` на том `/dev/sda2` при условии, что на `/dev/sda2` достаточно свободного места.

Резервное копирование одного из томов жёсткого диска выполняется аналогично:

```
# dd if=/dev/sda bs=1024k of=/media/storage/backup
```

Можно резервировать как жёсткий диск целиком, так и отдельный раздел (например, `/dev/sda2`).

П р и м е ч а н и е. Резервировать или восстанавливать том или жёсткий диск, на котором находится работающая в данный момент ОС, нельзя. Для резервного копирования системных файлов используется метод, описанный в разделе 7.9.2.

Для экономии места копию диска сжимают архиватором:

```
# dd if=/dev/sda bs=1024k | gzip > /media/storage/backup.gz
```

Универсальный метод, позволяющий «на лету» сохранять резервную копию тома на удалённой машине — использование протокола SSH:

```
# dd if=/dev/sdc1 | gzip | ssh user@hostname 'cat > backup.gz'
```

Длительность копирования зависит от размера исходного файла.

Восстановление из резервной копии выполняется командой:

```
# dd of=/dev/sda bs=1024k if=/media/storage/backup
```

Или, из сжатой резервной копии:

```
# gunzip -c /media/storage/backup.gz | dd of=/dev/sda bs=1024k
```

В резервный файл копируется полностью жесткий диск, структура, расположение файлов и свободное пространство. Такой метод копирования удобен, если нужно создать образ жесткого диска с несколькими разделами, что позволяет, например, устанавливать операционную систему сразу на множество машин путем копирования заранее созданного образа или восстанавливать работоспособность после замены жёсткого диска на полностью аналогичный.

7.9.2. Резервное копирование файлов

Создание резервных копий файлов выполняется для важных данных при этом копируются только данные, которые нужно сохранить. Для этих целей подходит программа ленточного архивирования: `tar`. Резервный файл сжимается архиватором `gzip` или `bzip2`.

Создание резервной копии с использованием программы `tar`:

```
# cd /
# tar -zcvpf /media/storage/home.bak.tar.gz /home
```

Параметр `-z` может быть опущен и используется для сжатия полученного архива.

Восстановление из резервной копии:

```
# cd /
# tar -zxvpf /media/storage/home.bak.tar.gz
```

Операции создания резервных копий автоматизируются с помощью Bash-сценариев. Разработанные сценарии операций по резервному копированию данных добавляются в планировщик заданий `cron`.

Рекомендуется предварительно тестировать сценарии сохранения/восстановления файлов, прежде чем использовать по назначению. Перед использованием сценариев рекомендуется сделать ещё одну резервную копию.

7.9.3. Резервное копирование при помощи `rsync`

Утилита `rsync` — гибкое средство резервного копирования. Она использует алгоритм поиска изменений, что позволяет пересылать только те данные, которые изменились.

Формат команды:

```
rsync [<параметры>] <исходный путь> <конечный путь>
```

7.9.3.1. Варианты использования

Относительно `rsync` существует восемь способов его использования:

1) для локального копирования файлов, когда ни исходный, ни конечный пути не содержат двоеточия «:»;

2) для копирования локальных файлов на удаленный хост, используя программу удаленной оболочки в качестве транспорта (например, SSH). В этом случае конечный путь содержит одно двоеточие «:» как разделитель адреса удаленного хоста и пути файловой системы на нём;

3) для копирования с удаленного хоста в локальные файлы, используя программу удаленной оболочки. Это происходит, если исходный путь содержит двоеточие «:»;

4) для копирования с удаленного `rsync`-сервера в локальные файлы, когда исходный путь содержит либо двойное двоеточие «::», либо он сформирован в виде URL, начинающийся с `rsync://`;

5) для копирования локальных файлов на удаленный `rsync`-сервер. В этом случае конечный путь содержит «::» или является URL вида `rsync://`;

6) для копирования с удаленной машины с использованием удаленной оболочки как транспорта и удаленного `rsync`-сервера. Это происходит, когда исходный путь содержит разделителем двойное двоеточие «::» и установлен параметр `--rsh=COMMAND;`

7) для копирования с локальной машины на удаленную с использованием удаленной оболочки как транспорта и удаленного `rsync`-сервера. Это происходит, когда конечный путь содержит разделителем двойное двоеточие «::» и установлен параметр `--rsh=COMMAND;`

8) Для получения списка файлов на удаленной машине. Это происходит, если не указать (оставить пустым) локальный путь.

Для получения дополнительной информации об утилите `rsync` выполните следующую команду:

```
man rsync
```

7.9.3.2. Использование `rsync` как сервера

Реализация `rsync`-сервера происходит следующим образом. Имеется отдельный сервер, куда будут складываться все резервные копии. С этого сервера запускается (как правило, с использованием планировщика `cron`, см. 8.4.1) команда `rsync` с параметрами, которая реализует соединение к удалённым машинам в сети. На всех машинах работает `rsync`-служба, в настройках которой указано, какие именно каталоги нужно синхронизировать. Примерная схема резервирования изображена на рисунке 2.

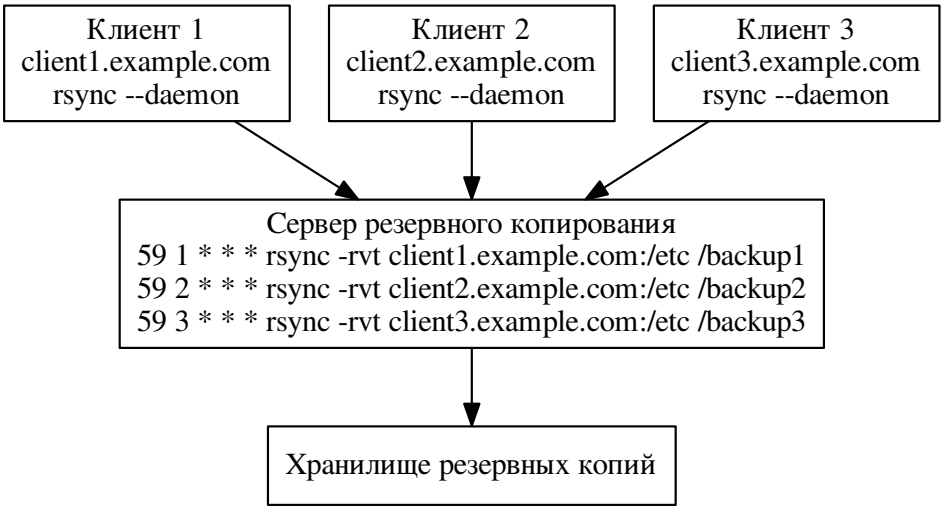

Рисунок 2 – Организация резервного копирования с помощью `rsync`-сервера

Следующие действия выполняются на каждом из клиентов (компьютеры, с которых нужно забирать резервные копии).

Файл с настройками rsync — /etc/rsyncd.conf.

Простейший файл настройки для экспорта каталога /etc/:

```
[etc]
    path = /etc/
```

На клиенте нужно настроить iptables. Для этого нужно добавить следующую строчку в файл /etc/sysconfig/iptables:

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 873 -j ACCEPT
```

Порт 873 — порт по умолчанию для rsync-сервера. После изменения списка правил нужно перезагрузить их:

```
# service iptables restart
```

Запуск сервера rsync осуществляется следующим образом:

```
# rsync --daemon
```

После чего станет возможным копирование каталога /etc/, данную команду нужно выполнять на сервере:

```
# rsync -rvt rsync://<IP>/etc etc.backup
```

Здесь <IP> — IP-адрес или домен клиента, с которого нужно выполнить резервную копию.

7.9.4. Резервное копирование баз данных

Допустимо выполнять резервное копирование файлов СУБД с данными при помощи методов, описанных в разделе «Резервное копирование файлов», но большинство современных СУБД обладают собственными средствами создания резервных копий. Для руководства необходимо смотреть соответствующие разделы документации по СУБД.

7.10. Работа с файлами подкачки

Подкачка страниц — механизм организации страничной виртуальной памяти в ОС. При нехватке свободного физического пространства в ОЗУ, страницы виртуальной памяти из ОЗУ перемещаются в область подкачки. При этом область подкачки может являться:

– отдельным разделом на жёстком диске (так называемый swap-раздел);

– файлом (файл подкачки);

– областью в оперативной памяти, в которую помещаются неиспользуемые страницы с использованием алгоритма сжатия данных.

Файл или раздел подкачки нельзя примонтировать, вместо этого для управления подкачкой используются команды `swapon` и `swapoff`.

8. ПРОЦЕССЫ И ЗАДАЧИ

Процессы играют ключевую роль в операционной системе. Базовую функциональность предоставляет ядро операционной системы в виде системных вызовов, а дополнительные функциональные возможности реализуют процессы. От набора процессов, выполняющихся в системе, зависит, какие задачи она выполняет: веб-сервер, клиент, база данных и т. п.

Процесс — это объект ядра операционной системы, который состоит из адресного пространства памяти и набора структур данных. По сути, процесс это запущенный экземпляр программы или службы. Программа не то же самое, что процесс: программа состоит из исполняемых файлов, объектных файлов, исходных текстов и ресурсов, но она может породить не один процесс. Так, например, веб-сервер запускает не менее двух процессов: один для обработки входящих соединений, а другой — для обработки пользовательских запросов.

Каждый запущенный процесс может породить другие процессы — потомки. Процесс, запустивший новый процесс называется родительским процессом. Новый процесс по отношению к создавшему его процессу называется дочерним.

Каждый процесс характеризуется набором атрибутов, который отличает данный процесс от всех остальных процессов. К таким атрибутам относятся:

1) идентификатор процесса (PID). Каждый процесс в системе имеет уникальный идентификатор. Каждый новый запущенный процесс получает номер на единицу больше предыдущего;

2) идентификатор родительского процесса (PPID). Данный атрибут процесс получает во время своего запуска и используется для получения статуса родительского процесса;

3) реальный и эффективный идентификаторы пользователя (UID, EUID) и группы пользователя (GID, EGID). Данные атрибуты процесса говорят о его принадлежности к конкретному пользователю и группе. Реальные идентификаторы совпадают с идентификаторами пользователя, который запустил процесс, и группы, к которой он принадлежит. Эффективные — от чьего имени был запущен процесс. Права доступа процесса к ресурсам определяются эффективными идентификаторами. Если на исполняемом файле программы установлен специальный бит SGID или SUID, то процесс данной программы будет обладать правами доступа владельца исполняемого файла. Для управления процессом используются реальные идентификаторы. Все идентификаторы передаются от родительского процесса к дочерне-

му. Для просмотра данных атрибутов можно воспользоваться командой `ps`, задав желаемый формат отображения;

4) приоритет или динамический приоритет (`priority`) и относительный или статический (`nice`) приоритет процесса. Статический приоритет или nice-приоритет лежит в диапазоне от минус 20 до плюс 19, по умолчанию используется значение 0. Значение минус 20 соответствует наиболее высокому приоритету, nice-приоритет не изменяется планировщиком, он наследуется от родительского процесса или его указывает пользователь вручную. Динамический приоритет используется планировщиком для планирования выполнения процессов. Этот приоритет хранится в поле `prio` структуры `task_struct` процесса. Динамический приоритет вычисляется исходя из значения параметра `nice` для данной задачи путем вычисления надбавки или штрафа, в зависимости от интерактивности задачи. Пользователь имеет возможность изменять только статический приоритет процесса и только в меньшую сторону, повышать приоритет может только системный администратора. Управления приоритетом процессов осуществляется командами `nice` и `renice;`

5) состояние процесса. В ОС Linux каждый процесс обязательно находится в одном из перечисленных ниже состояний и может быть переведен из одного состояния в другое системой или командами пользователя. Различают следующее состояния процессов:

– `TASK_RUNNING` — процесс готов к выполнению или выполняется (runnable). Обозначается символом `R;`

– `TASK_INTERRUPTIBLE` — ожидающий процесс (sleeping). Данное состояние означает, что процесс инициализировал выполнение какой-либо системной операции и ожидает её завершения. К таким операциям относятся ввод/вывод, завершение дочернего процесса и т. д. Процессы с таким состоянием обозначаются символом `S;`

– `TASK_STOPPED` — выполнение процесса остановлено (stopping). Любой процесс можно остановить. Это может делать как система, так и пользователь. Состояние такого процесса обозначается символом `T;`

– `TASK_ZOMBIE` — завершившийся процесс (Зомби). Процессы данного типа возникают в случае, когда родительский процесс не ожидая завершения дочернего процесса, продолжает параллельно работать. Процессы с таким состоянием обозначаются символом `Z`. Завершившиеся процессы больше не

выполняются системой, но по-прежнему продолжают потреблять её невычислительные ресурсы. Убить (отправить сигнал SIGKILL) зомби нельзя;

– TASK_UNINTERRUPTIBLE — непрерываемый процесс (uninterruptible). Процессы в данном состоянии ожидают завершения операции ввода/вывода с прямым доступом в память. Такой процесс нельзя завершить, пока не завершится операция ввода/вывода. Процессы с таким состоянием обозначаются символом D. Состояние аналогично TASK_INTERRUPTIBLE, за исключением того, что процесс не возобновляет выполнение при получении сигнала. Используется в случае, когда процесс должен ожидать беспрерывно или когда ожидается, что некоторое событие может возникать достаточно часто. Так как задача в этом состоянии не отвечает на сигналы, TASK_UNINTERRUPTIBLE используется менее часто, чем TASK_INTERRUPTIBLE.

Типы процессов:

1) системные процессы — являются частью ядра и всегда расположены в оперативной памяти. Системные процессы не имеют соответствующих им программ в виде исполняемых файлов и запускаются при инициализации ядра системы. Выполняемые инструкции и данные этих процессов находятся в ядре системы, таким образом, они могут вызывать функции и обращаться к данным, недоступным для остальных процессов. Системными процессами, например, являются: shed (диспетчер свопинга), vhand (диспетчер страничного замещения), kmadaemon (диспетчер памяти ядра);

2) службы — это неинтерактивные процессы, которые запускаются обычным образом — путем загрузки в память соответствующих им программ (исполняемых файлов), и выполняются в фоновом режиме. Обычно службы запускаются при инициализации системы (но после инициализации ядра) и обеспечивают работу различных подсистем: системы терминального доступа, системы печати, системы сетевого доступа, почтовый сервер, dhcp-сервер и т. п. Службы не связаны ни с одним пользовательским сеансом работы и не могут непосредственно управляться пользователем. Большую часть времени демоны ожидают, пока тот или иной процесс запросит выполнить определённое действие, например, доступ к файловому архиву или печать документа;

3) прикладные (пользовательские) процессы — к прикладным процессам относятся все остальные процессы, выполняющиеся в системе. Как правило, это процессы, порождённые в рамках пользовательского сеанса работы. Например, ко-

манда `ls` породит соответствующий процесс этого типа. Важнейшим прикладным процессом является командный интерпретатор (Bash), который обеспечивает вашу работу в командной строке. Он запускается сразу же после регистрации в системе. Прикладные процессы могут выполняться как в интерактивном, так и в фоновом режиме, но в любом случае время их жизни (и выполнения) ограничено сеансом работы пользователя. При выходе из системы все прикладные процессы будут уничтожены.

Процессы подчинены чёткой иерархии. Каждый процесс в системе имеет всего одного родителя и может иметь один или более порождённых процессов.

На последней фазе загрузки ядро монтирует корневую файловую систему и формирует среду выполнения нулевого процесса, создавая пространство процесса, инициализируя нулевую точку входа в таблице процесса и делая корневой каталог текущим для процесса. Когда формирование среды выполнения процесса заканчивается, система исполняется уже в виде нулевого процесса. Нулевой процесс «ветвится», запуская `fork` прямо из ядра, поскольку сам процесс исполняется в режиме ядра. Код, исполняемый порождённым процессом с номером 1, включает в себя вызов системной функции `exec`, запускающей на выполнение программу `systemd`. В отличие от нулевого процесса, который является процессом системного уровня, выполняющимся в режиме ядра, процесс `systemd` относится к пользовательскому уровню. Процесс `systemd` отвечает за инициализацию новых процессов.

8.1. Утилиты управления процессами

Возможны следующие операции над процессами:

– порождение процесса (запуск программы);

– отправка процессу сигнала, в том числе сигнала завершения и сигнала `SIGKILL`;

– изменение приоритета.

Также присутсвует возможность мониторинга процессов, описанная в разделе 9.1.

8.1.1. `kill`

Команда `kill` завершает процесс или отправляет ему сигнал по идентификатору процесса.

Для каждого операнда PID команда `kill` будет отправлять указанный сигнал.

Формат команды:

```
kill -l [<статус выхода>]
```

или

```
kill [ -<сигнал> ] <PID...>
```

или

```
kill -s <сигнал> <PID...>
```

Процесс, которому посылается сигнал, должен принадлежать текущему пользователю, но системный администратор может посылать сигналы любым процессам.

Параметры.

`-l [<статус выхода>]` Выдать все значения сигналов, поддерживаемые в данной реализации, если операнды не указаны. Если указан операнд `<статус выхода>`, соответствующий значению специального параметра ? командного интерпретатора и вызову `wait` для процесса, работа которого прекращена сигналом, выдать имя сигнала, которым была прекращена работа процесса. Если указан операнд `<статус выхода>`, представляющий собой целое число без знака — номер сигнала, — выдать имя соответствующего сигнала. В противном случае, результат не определен.

`-<сигнал> или -s <сигнал>` Задает сигнал, который надо послать, используя одно из символьных имен, заданных в заголовочном файле `<signal.h>`. Значение сигнала распознаётся независимо от регистра символов. При этом префикс `SIG` указывать не надо. Кроме того, распознается символьное имя 0, представляющее сигнал со значением ноль. По умолчанию отправляется сигнал `SIGTERM`. Список сигналов приведён ниже.

Список допустимых сигналов приведён в таблице 13.

Таблица 13 – Описание системных сигналов

Сигнал	Номер	Описание
`SIGHUP`	1	Закрытие терминала
`SIGINT`	2	Сигнал прерывания [Ctrl+C] с терминала
`SIGQUIT`	3	Сигнал «Выход» с терминала [Ctrl+\]
`SIGILL`	4	Недопустимая инструкция процессора
`SIGTRAP`	5	Ловушка трассировки или брейкпоинт
`SIGABRT`	6	Сигнал, посылаемый функцией `abort()`

Сигнал	Номер	Описание
SIGBUS	7	Неправильное обращение в физическую память
SIGFPE	8	Ошибочная арифметическая операция
SIGKILL	9	Безусловное завершение
SIGUSR1	10	Пользовательский сигнал № 1
SIGSEGV	11	Нарушение при обращении в память
SIGUSR2	12	Пользовательский сигнал № 2
SIGPIPE	13	Запись в разорванное соединение (пайп, сокет)
SIGALRM	14	Сигнал истечения времени, заданного alarm()
SIGTERM	15	Сигнал завершения (сигнал по умолчанию для утилиты kill)
SIGSTKFLT	16	Ошибка обращения в стек
SIGCHLD	17	Дочерний процесс завершен или остановлен
SIGCONT	18	Продолжить выполнение ранее остановленного процесса
SIGSTOP	19	Остановка выполнения процесса
SIGTSTP	20	Сигнал остановки с терминала [Ctrl+Z]
SIGTTIN	21	Попытка чтения с терминала фоновым процессом
SIGTTOU	22	Попытка записи на терминал фоновым процессом
SIGURG	23	На сокете получены срочные данные
SIGXCPU	24	Процесс превысил лимит процессорного времени
SIGXFSZ	25	Процесс превысил допустимый размер файла
SIGVTALRM	26	Истечение «виртуального таймера»
SIGPROF	27	Истечение таймера профилирования
SIGWINCH	28	Изменение размера окна
SIGPOLL	29	Событие, отслеживаемое poll()
SIGPWR	30	Ошибка питания
SIGSYS	31	Неправильный системный вызов

Следующие примеры эквивалентны и отправляют SIGKILL процессу с PID = 100:

```
kill -9 100
kill -s kill 100
kill -s KILL 100
kill -KILL 100
```

Сигнал SIGKILL

Сигнал SIGKILL имеет особое значение. Будучи посланным процессу, SIGKILL вызывает его немедленное завершение. В отличие от SIGTERM или SIGINT этот сигнал не может быть перехвачен или проигнорирован, а процесс, получивший его, не имеет возможности выполнить какие-либо действия перед своим завершением.

Данный сигнал нужно использовать только в случае крайней необходимости. Иногда процессам нужно несколько секунд или минут, чтобы завершить свою работу, они могут выполнять после завершения важную работу, например, сохранять данные. Если в этот момент отправить SIGKILL, данные будут безвозвратно утеряны.

Особенности сигнала SIGKILL:

– процесс-зомби нельзя завершить SIGKILL, потому что зомби уже завершен и не может принимать сигналов. Зомби ожидает, что родительский процесс считает код завершения с помощью системного вызова wait();

– процессы в состоянии блокировки (например, при ожидании ввода/вывода) не смогут завершиться SIGKILL, пока операционная система не вернет их в нормальное состояние (при наступлении ожидаемого события или ошибке);

– процесс init является особым случаем — он не получает от операционной системы сигналов, которые он не хочет обрабатывать, и, следовательно, может игнорировать SIGKILL.

8.1.2. killall

Команда killall аналогична команде kill за исключением, что в качестве аргумента принимает имя команды, а не ей идентификационный номер, что значительно упрощает поиск процесса, которому нужно отправить сигнал. Если по данному имени найдено несколько процессов, сигнал будет отправлен им всем, поэтому необходимо осторожно выполнять данную команду, чтобы случайно не завершить важный процесс.

Аргументы аналогичны команде kill.

8.1.3. sleep

Команда sleep « засыпает» на определённое время. Основное применение — выдержка паузы в Bash сценариях.

Формат команды:

```
sleep [<длительность в секундах>]
```

По умолчанию, вызванная без аргументов, sleep засыпает на одну секунду.

Примеры.

Выводит «1», затем спустя одну секунду «2»:

```
$ echo 1; sleep; echo 2
```

Выполнить команду ls спустя две секунды в отдельном потоке (многопоточное выполнение в терминале с таймером):

```
$ sleep 2 && ls &q
```

Последняя команда очень удобна, когда в сценарии нужно выполнить новый поток с заданной задержкой.

8.1.4. usleep

Данная команда аналогична команде sleep за исключением того, что засыпает на указанное число микросекунд, что позволяет более точно контролировать время выполнения команд.

Например, задержка в одну секунду:

```
$ usleep 1000000
```

8.1.5. nice

Команда nice запускает программу с заданием приоритета. Без указания команды выдает текущий приоритет работы. Диапазон приоритетов расположен от минус 20 (наивысший приоритет) до плюс 19 (наименьший).

Формат команды:

```
nice [<параметры>] -<приоритет> [<команда с аргументами>]
```

Параметр -<приоритет> или -n <приоритет> или --ajustment=<приоритет> устанавливает приоритет.

Примеры:

```
$ nice -2 ls
$ nice -n 2 ls -l
```

```
$ nice --4 ls //отрицательное значение приоритета недоступно пользователю
nice: невозможно установить значение nice: Отказано в доступе
```

8.1.6. ionice

Изменить приоритет ввода-вывода процесса или команды.

Формат команды:

```
ionice [<параметры>] -p <PID...>
```

или

```
ionice [<параметры>] <команда>
```

Данная утилита позволяет изменить приоритет дисковых операций как уже работающего процесса по его идентификатору (PID) — первый вариант, так и запустить команду с заданным приоритетом ввода-вывода (второй вариант). Чаще всего используется для запуска фоновых операций с пониженным приоритетом.

Параметры.

`-n N` Задать приоритет данных от 0 — самый высокий до 7 — самый низкий.

`-c N` Задать класс планировщика: 0 — по умолчанию, 1 — реального времени, 2 — максимально быстрый, 3 — фоновый.

8.1.7. renice

Изменить приоритет уже работающего процесса. Можно изменить приоритет по идентификатору, имени пользователя или группе. Одновременно можно изменить приоритет нескольким процессам.

Формат команды:

```
renice <приоритет> [[-p] <PID...>] [[-u] <пользователь...>] [[-g] <группа...>]
```

Параметры.

`-p` Рассматривать последующие аргументы как идентификаторы процессов.

`-u` Рассматривать последующие аргументы как имена или идентификаторы пользователей.

`-g` Рассматривать последующие аргументы как имена или идентификаторы групп.

Приоритет можно задать как конкретным числом от -20 до 19, так и инкрементом/декрементом текущего приоритета, например:

– +2 — увеличить приоритет на 2;

– -3 — уменьшить приоритет на 3;

– 2 — установить приоритет в 2.

Пример:

```
$ renice +1 -u 500 root daemon -p 992
```

Инкремент приоритета на единицу для процессов пользователя с идентификатором 500, `root`, `daemon` и процесса с номером 992.

8.1.8. `pgrep`

Команда `pgrep` предназначена для поиска процессов по различным критериям, результат формируется в виде идентификаторов процессов.

Формат команды:

```
pgrep [<параметры>] <шаблон...>
```

Параметры.

`-d <разделитель>` Задаёт строку-разделитель результатов, которая выдается между идентификаторами соответствующих процессов. Если параметр `-d` не указан, выдается символ новой строки. Параметр `-d` можно задавать только утилите `pgrep`.

`-f` Шаблону (который задается как регулярное выражение) должна соответствовать полная строка аргументов процесса. Если параметр `-f` не указан, шаблону должно соответствовать только имя выполняемого файла.

`-g <список групп>` Выбирает только процессы, идентификатор группы которых входит в заданный список.

`-G <список групп>` Выбирает только процессы, реальный идентификатор группы которых входит в заданный список. В качестве идентификатора группы можно задавать либо ее имя, либо её числовой идентификатор.

`-l` «Длинный» формат выдачи результатов. Для каждого выбранного процесса помимо идентификатора выдает имя. Параметр `-l` можно задавать только для утилиты `pgrep`.

`-n` Выбирает только самый новый (созданный последним) процесс, удовлетворяющий остальным критериям.

`-P <список ppid>` Выбирает только процессы, идентификатор родительского процесса которых входит в указанный список.

`-s <список sid>` Выбирает только процессы, идентификатор сеанса которых входит в заданный список.

`-t <список терминалов>` Выбирает только процессы, связанные с одним из терминалов в указанном списке.

`-T <список taskid>` Выбирает только процессы, идентификатор задачи которых входит в заданный список.

`-u <список euid>` Выбирает только процессы, эффективный идентификатор пользователя которых входит в указанный список.

`-U <список uid>` Выбирает только процессы, реальный идентификатор пользователя которых входит в указанный список.

`-v` Обращает критерии выбора процессов — выбирает все процессы, кроме удовлетворяющих заданным критериям поиска.

`-x` Выбирает только процессы, строка аргументов или имя выполняемого файла которых точно соответствует заданному шаблону. Соответствие шаблону считается точным, когда все символы в строке аргументов процесса или имени выполняемого файла соответствуют шаблону.

Примеры.

Выдаёт процессы, реальный идентификатор группы которых имеет значение `other` или `daemon`:

```
$ pgrep -G other,daemon
```

Если указано несколько критериев поиска, утилита pgrep ищет процессы, атрибуты которых соответствуют всем критериям.

Эта команда выдает процессы, у которых реальный идентификатор группы — `other` или `daemon` и реальный идентификатор пользователя — `root` или `daemon`:

```
$ pgrep -G other,daemon -U root,daemon
```

8.1.9. `pkill`

Утилита `pkill` работает аналогично `pgrep`, но каждому соответствующему критериям поиска процессу посылается сигнал, аналогично `kill`, вместо выдачи идентификатора процесса. Имя или номер сигнала нужно указать как первый параметр командной строки `pkill`.

Формат команды:

```
pkill [<сигнал>] [<параметры>] <критерий...>
```

Критерии и параметры полностью аналогичны `pgrep`, сигналы полностью аналогичны параметрам команды `kill`.

Примеры.

Завершить работу последнего запущенного окна `xterm`:

```
$ pkill -n xterm
```

Завершить работу «зависшего» процесса с именем `Thunar`:

```
$ pkill -9 Thunar
```

8.1.10. `fuser`

`fuser` — утилита, идентифицирующая процессы, которые используют указанные файлы или сокеты.

Формат команды:

```
fuser [-a|-s|-c] [-4|-6] [-n <пространство>] [-k [-i] [-<сигнал>] ] [-muvf] <имя...>
```

или

```
fuser -l
```

или

```
fuser -V
```

`fuser` отображает идентификаторы процессов (PID), которые используют в данный момент указанные файлы или файловые системы. По умолчанию, во время вывода информации после имени каждого файла следует буква, показывающая вид доступа:

– `c` — текущий каталог;

– `e` — запущенный исполняемый файл;

– `f` — открытый файл. По умолчанию в режиме вывода информации буква `f` может отсутствовать;

– `F` — файл открыт для записи;

– `r` — корневой каталог;

– `m` — файл является отображаемым или разделяемой (совместно используемой) библиотекой.

Команда `fuser` возвращает ненулевой код возврата, если никакой из указанных файлов не «захвачен» ни одним процессом или в случае возникновения фаталь-

ной ошибки. Если найден хотя бы один процесс, который использует <имя>, тогда команда fuser возвращает ноль.

В случае просмотра процессов, которые используют сокеты TCP и UDP, соответствующее название <пространство> должно быть указано с параметром -n. По умолчанию, команда fuser будет просматривать оба сокета IPv6 и IPv4. Чтобы изменить установки, действующие по умолчанию, необходимо использовать параметры -4 и -6. В качестве сокета(ов) может быть указан как локальный, так и удалённый порт или удалённый адрес. Все поля являются необязательными, однако перед пропущенными полями запятые должны присутствовать:

```
[lcl_port][,[rmt_host][,[rmt_port]]]
```

Для IP-адресов и номеров портов может быть указан либо номер порта, либо его символьное название.

На стандартное устройство вывода fuser направляет только идентификаторы процессов (PID), всё остальное направляется на стандартное устройство вывода ошибок.

Параметры.

-a Показывает информацию для всех файлов, которые указаны в командной строке. По умолчанию, выводятся имена только тех файлов, которые используются хотя бы одним процессом.

-c Подобно параметру -m и применяется для совместимости с POSIX.

-f Игнорируется без предупреждения. Применяется для совместимости с POSIX.

-k Уничтожает (завершает) процессы, которые используют указанный файл. Посылаемый сигнал завершения SIGKILL можно заменить с помощью параметра -signal. Процесс fuser никогда не уничтожит себя сам, однако может завершить работу других процессов fuser. Перед попыткой уничтожить выполняющийся процесс, команда fuser устанавливает эффективный идентификатор пользователя ID этого процесса в идентификатор собственного пользователя.

-i Перед завершением процесса потребует подтверждения от пользователя. Если не задан параметр -k, этот параметр игнорируется без предупреждения.

-l Выводит список всех существующих названий сигналов.

-m В качестве name указывается файл на смонтированной файловой системе или смонтированное специальное блочное устройство. Выводится список всех процессов, которые используют файлы на этой файловой системе. Если указанный

файл является каталогом, тогда к его имени автоматически добавляется name и рассматривается любая файловая система, которая может быть смонтирована на этот каталог.

-n <пространство> Выбирает различные множества имён. Поддерживаются такие множество имён как file (по умолчанию это имена файлов), udp (локальные порты UDP) и tcp (локальные порты TCP). Для портов может быть указан либо номер порта, либо его символьное название. Можно использовать сокращённую запись цифрами name/<пространство> (например, 80/tcp), если она однозначно характеризует объект.

-s Выполняет операции без вывода сообщений. Параметры -u и -v игнорируются в этом режиме. Параметр -a не должен использоваться с параметром -s.

-signal Посылает процессу указанный сигнал завершения работы вместо обычного SIGKILL. Сигналы могут быть заданы по названию (например, -HUP) или по номеру (например, -1). Этот параметр без предупреждения игнорируется, если не используется параметр -k.

-u Добавляет к каждому PID имя владельца процесса.

-v Режим подробного информирования. Процессы показываются в стиле вывода команды ps. Поля PID, USER и COMMAND подобны выводу команды ps. Поле ACCESS показывает процесс, который использует файл. Если объект используется ядром (например, в случае точек монтирования, swap файла и др.), вместо PID отображается строка kernel.

-4 Выполняется поиск только для сокета IPv4. Этот параметр не должен использоваться с параметром -6 и работает только с названиями space tcp и udp.

-6 Выполняется поиск только для сокета IPv6. Этот параметр не должен использоваться с параметром -4 и работает только с названиями space tcp и udp.

- Сбрасывает все параметры и устанавливает сигнал уничтожения процессов в SIGKILL.

Примеры.

Уничтожает все процессы, использующие каким-либо образом файловую систему /home:

```
$ fuser -km /home
```

Вызывает выполнение something, если никакой другой процесс не использует /dev/ttyS1:

```
$ if fuser -s /dev/ttyS1; then :; else something; fi
```

Показывает все процессы (локальные) на порте TELNET.

```
$ fuser telnet/tcp
```

8.1.11. lsof

lsof — утилита, служащая для вывода информации о том, какие файлы используются теми или иными процессами.

Примеры.

Просмотр всех соединений IPv4, открытых процессом с PID=1234:

```
$ lsof -i 4 -a -p 1234
```

Просмотр информации о процессе, который прослушивает 80 TCP порт:

```
$ lsof -i tcp:80
```

Список открытых файлов на устройстве /dev/hd4:

```
$ lsof /dev/hd4
```

Список процессов, работающих с компакт-диском:

```
$ lsof /dev/cdrom
```

Список подключений по ssh:

```
$ lsof -c ssh
```

8.2. Управление задачами

Задачи выполняются из командной оболочки и являются её элементами. Запуск задачи осуществляется путём ввода символа амперсанда — «&» после имени команды и её аргументов.

Когда вы запускаете команду, ваш текущий командный процессор определяет ее как задачу и следит за ней. Когда команда выполнена, соответствующая задача исчезает. Задачи находятся на более высоком уровне, чем процессы; операционная система ничего о них не знает. Они являются всего лишь элементами командного процессора.

Некоторые важные термины:

1) интерактивное задание (foreground job) — выполняемое в командном процессоре, занимающее сеанс командного процессора, так что вы не можете выполнить другую команду;

2) фоновое задание (background job) — выполняемое в командном процессоре, но не занимающее сеанс командного процессора, так что вы можете выполнить другую команду в этом же командном процессоре;

3) приостановить (suspend) — временно приостановить интерактивный процесс;

4) возобновить (resume) — вернуться к выполнению приостановленной задачи. В фон удобно отправлять задачи, не требующие вмешательства пользователя. Примерами таких задач могут служить компиляция программного обеспечения и сложные вычислительные программы. Для этих приложений важно минимизировать суммарное время выполнения в системе, загруженной другими процессами, порожденными, в частности, интерактивными задачами.

Если вы запустили из командного процессора команду в интерактивном режиме и хотите немедленно прекратить выполнение команды, введите [Ctrl+C]. Командный процессор воспримет нажатие [Ctrl+C] как «остановить выполнение текущей задачи немедленно». Поэтому, если вы выводите очень длинный файл (например, командой `cat`) и хотите остановить вывод, нажмите [Ctrl+C]. На самом деле текущей задаче по нажатию [Ctrl+C] отправится сигнал `SIGINT` (см. 8.1.1).

Чтобы прекратить выполнение программы, работающей в фоновом режиме, вы можете перевести ее в интерактивный режим с помощью команды `fg` и затем нажать [Ctrl+C], или использовать команду `kill`.

Ниже перечислены команды управления заданиями. Они не являются самостоятельными командами, а являются частью командной оболочки. Идентификатор задания можно узнать из вывода командной оболочки — он обозначается в квадратных скобках.

8.2.1. `jobs`

`jobs` — получение информации по задачам.

Формат команды:

```
jobs [-lnprs] [<задание...>]
```

Параметры.

`-l` Помимо обычной информации выдаёт идентификаторы процессов.

`-p` Выдает только идентификаторы процессов-лидеров групп процессов, образующих задание.

`-n` Выдает информацию только о заданиях, состояние которых изменилось с момента последнего уведомления пользователя о состоянии заданий.

`-r` Выдает только выполняющиеся задания.

`-s` Выдает только остановленные задания.

Если указано задание, выдается информация только об этом задании.

Статус выхода — 0, кроме случаев, когда указана недопустимый параметр или идентификатор несуществующего задания.

Если указан параметр `-x`, команда `jobs` заменяет любой идентификатор задания в команде или аргументах соответствующим идентификатором группы процессов, и выполняет команду, передавая ей аргументы и возвращая ее статус выхода.

8.2.2. `bg`

Возобновляет выполнение указанного задания в фоновом режиме, как если бы оно было запущено с конструкцией «&». Если задание не указано, в фоновый режим переводится текущее задание командного интерпретатора. Команда `bg` возвращает 0, если только управление заданиями не отключено или, при включённом управлении заданиями, соответствующее задание не найдено или запускалось при отключенном управлении заданиями.

Формат команды:

```
bg [<задание>]
```

8.2.3. `fg`

Возобновляет работу задания в приоритетном режиме и делает это задание текущим. Если задание не указано, используется текущее задание командного интерпретатора. Возвращается значение статуса выхода команды, переведённой в приоритетный режим, или 1, если управление заданиями отключено или, при включенном управлении заданиями, если указано несуществующее задание или задание, запущенное при отключённом управлении заданиями.

Формат команды:

```
fg [<задание>]
```

Пример:

1) открыть редактор `vi`:

```
# vi /etc/passwd
```

2) перевести редактор из интерактивного режима в фоновый: нажать сочетание клавиш [Ctrl+Z];

3) выполнить команду `ls`:

```
# ls
```

4) вернуть редактор `vi` из фонового режима в интерактивный и продолжить работу:

```
# fg
```

8.2.4. `suspend`

Приостанавливает работу текущего командного интерпретатора, пока он не получит сигнал `SIGCONT`. Параметр `-f` отключает предупреждения, если эта команда выполняется в начальном командном интерпретаторе — безусловно приостановить работу. Статус выхода — 0, если только командный интерпретатор не является начальным, а параметр `-f` не указана, или если управление заданиями отключено.

Формат команды:

```
suspend [-f]
```

8.2.5. stop

Останавливает выполнение фонового задания.

Формат команды:

```
stop <задание>
```

8.2.6. wait

Ожидает завершения выполнения задания и возвращает код возврата задания.

Формат команды:

```
wait <задание>
```

8.3. Службы ОС

Службы в операционной системе выполняют различные задачи: планирование запуска процессов, обеспечение поддержки сети, поддержка системы печати, почтовый сервер, веб-сервер и т. п. Управление службами отличается от управления обычными пользовательскими процессами.

Рассмотрим наиболее важные службы операционной системы:

1) планировщик заданий `crond` — запускает программы или сценарии с заданными интервалами времени;

2) система регистрации событий — `auditd`;

3) служба инициализация сети network — набор сценариев, который производит настройку интерфейсов, маршрутизации;

4) служба инициализации межсетевого экрана iptables. Сам iptables является лишь интерфейсом к модулю ядра операционной системы netfilter и используется для его настройки. Для сетей IPv6 используется ip6tables. Не является демоном;

5) сервер удалённого входа в систему sshd;

6) служба тестирования операционной системы — ztestd;

7) сетевой сервер системы печати — cups. cups осуществляет печать документов как для локальных пользователей, так и удалённых пользователей через сеть.

8.3.1. Управление системными службами

Системные службы — это программы, работающие в фоновом режиме и выполняющие в системе определённые функции.

Для обращения к службе используется команда service. В качестве первого аргумента команде подаётся имя службы, в качестве второго — параметр запуска. В общем случае управление службами происходит следующим образом:

```
service <имя службы> <параметр запуска>
```

Возможны следующие параметры запуска:

– start — запуск;

– stop — остановка;

– restart — перезапуск;

– status — отображает состояние службы, для большинства служб данный параметр лишь сообщает о том, работает ли демон, но некоторые, например, iptables, выдают другую полезную информацию.

Таким образом, каждая служба может быть остановлена, запущена или перезапущена. Однако, каждая служба может также иметь некоторые дополнительные параметры.

Например, для перезапуска службы системного журнала необходимо выполнить команду:

```
# service network restart
```

Команда service позволяет управлять службами в режиме реального времени, но при этом после перезагрузки системы конфигурация запуска системных

служб будет восстановлена по умолчанию. Для того, чтобы установить запуск какой-либо службы одновременно со стартом системы, нужно использовать команду chkconfig. chkconfig позволяет включать и отключать автоматический запуск служб при старте операционной системы. Формат команды:

```
chkconfig <имя службы> on|off
```

Если задано значение on, служба будет запускаться при старте системы, и не будет запускаться, если было передано значение off. Данные параметры сохраняются и используются при последующих загрузках.

Любое управление службами должен осуществлять системный администратор. В большинстве случаев, системные службы настроены по умолчанию и запускаются при старте операционной системы, поэтому не требуют каких-либо дополнительных действий по их регулированию. Перезапуск некоторых демонов возможен для обновления параметров конфигурации.

8.4. Планирование задач

8.4.1. cron

Для автоматического регулярного выполнения задач используется программа cron. Запуск программ по расписанию происходит на основе конфигурационных файлов, называемых crontab. В каталоге /var/spool/cron/ хранятся все crontab-файлы, но редактировать вручную их не рекомендуется, для этого существует специальная программа crontab.

Конфигурационный файл crontab состоит из строк, в каждой строке располагается задание. В строке кодируется расписание, когда выполнять команду и сама команда.

Формат строки-задания:

```
* * * * * выполняемая команда
- - - - -
| | | | |
| | | | ----- День недели (0-7) (Воскресенье =0 или =7)
| | | ------- Месяц (1 - 12)
| | --------- День (1 - 31)
| ----------- Час (0 - 23)
------------- Минута (0 - 59)
```

Хорошей практикой является написание администратором собственных сценариев администрирования и обслуживания с последующим их добавлением в crontab.

Для редактирования конфигурационного файла в cron используется команда crontab. Эту команду нужно запускать от имени пользователя, от которого должны выполняться запланированные задачи.

Для редактирования своего crontab, выполните команду:

```
$ crontab -e
```

Откроется персональный файл с заданиями для текущего пользователя в редакторе vim (см. 4.8). Пользователь может указать явно, какой текстовый редактор использовать для редактирования crontab, передав его в переменную окружения EDITOR.

Пользователи могут устанавливать свои независимые crontab и им не нужно задумываться о том, в какой файл или каталог нужно сохранить задания.

Параметры команды crontab:

– -u <имя пользователя> — администратор указывает имя пользователя, чей crontab должен быть отредактирован, если параметр не указан, используется crontab пользователя, выполняющего команду;

– -l — отображает текущий файл crontab на стандартный вывод;

– -r — удаляет текущий файл crontab;

– -e — изменяет файл crontab редактором, указанным в VISUAL или в EDITOR переменной окружения, после выхода из редактора, измененный файл crontab будет автоматически установлен.

Пример конфигурационного файла crontab:

```
# с символа '#' начинаются комментарии

# в качестве интерпретатора использовать /bin/sh
SHELL=/bin/sh

# результаты работы отправлять по электронной почте
MAILTO=admin@example.org

# добавить в PATH каталог $HOME/bin
PATH=$PATH:$HOME/bin
```

```
#### Задания ####

# выполнять каждый день в 0 часов 5 минут, результат
складывать в log/daily
5 0 * * * $HOME/bin/daily.job >> $HOME/log/daily 2>&1
# выполнять 1 числа каждого месяца в 14 часов 15 минут
15 14 1 * * $HOME/bin/monthly
# каждый рабочий день в 22:00
0 22 * * 1-5 echo "Пора домой" | mail -s "Уже 22:00" john
23 */2 * * * echo "Выполняется в 0:23, 2:23, 4:23 и т. д."
5 4 * * sun echo "Выполняется в 4:05 в воскресенье"
0 0 1 1 * echo "С новым годом!"
15 10,13 * * 1,4 echo "Эта надпись выводится в понедельник и
четверг в 10:15 и 13:15"
0-59 * * * * echo "Выполняется ежеминутно"
# каждые 5 минут
*/5 * * * * echo "Прошло пять минут"
```

Для системного администрирования пользовательские `crontab` не годятся. Для автоматизации действий системного администратора необходимо создать задачу в системном `crontab`, который располагается в файле `/etc/crontab`.

Для разработчиков программ также присутствует специальный каталог `/etc/cron.d`, в который при установке программы нужно копировать файл с заданиями, а при удалении программы данный файл удаляется, тем самым удаляя ненужные более в системе задачи. Данный каталог преимущественно нужен для автоматической установки и удаления программ и для удобства администрирования.

Существует более простой и быстрый способ использовать планировщик для системного администрирования. Системный администратор может создать сценарий на языке Bash (см. 4.2), а затем скопировать его в один из следующих каталогов:

– `/etc/cron.hourly/` — для задач, которые нужно запускать каждый час;

– `/etc/cron.daily/` — для ежедневных задач;

– `/etc/cron.weekly/` — для еженедельных задач;

– `/etc/cron.monthly/` — для ежемесячных задач.

Таким образом можно быстро спланировать, какие периодически задания нужно запускать, так как ежедневные и ежемесячные задания являются наиболее часто используемыми.

8.4.2. anacron

anacron в отличие от cron не поддерживает запуск заданий по расписанию, вместо этого задания запускаются с заданным интервалом времени. Это очень удобно для систем которые работают не регулярно, например домашние рабочие станции или ноутбуки. Задачи anacron хранятся в файле конфигурации /etc/anacrontab. Задания выполняются даже в том случае, если в момент времени, когда нужно было запустить задачу, компьютер был выключен.

Формат файла:

```
* * * выполняемая команда
- - -
| | |
| | --------- идентификатор
| ----------- задержка
------------- период
```

Период — период выполнения в днях. Задержка — задержка запуска в минутах. Идентификатор задания — любой непустой символ, кроме «/» и «\». Задержка чаще всего используется для того чтобы позволить системе полностью загрузиться.

9. МОНИТОРИНГ И ЖУРНАЛИРОВАНИЕ

Мониторинг и журналирование событий — важнейшая задача администратора — не только в связи с поддержанием уровня безопасности системы, но для анализа неисправностей. Журналирование является нормой для всех служб в системе и присутствует в ОС.

9.1. Мониторинг системы и сбор статистики

Для сбора статистики в ОС предусмотрены специальные файлы, в которые записываются происходящие события.

9.1.1. Файловая система `/proc`

Файловая система `/proc` является механизмом для ядра и его модулей, позволяющим передавать данные процессам. С помощью файловой системы `/proc` осуществляется взаимодействие с внутренними структурами ядра, получение данных о процессах и передача установок параметров ядра на лету. Файловая система `/proc` располагается в памяти в отличие от других файловых систем, которые располагаются на диске.

С помощью файлов в `/proc` можно получать информацию о состоянии ядра, процессов, параметрах компьютера и т. д. Большинство файлов в `/proc` содержат самую свежую информацию о системном оборудовании. Несмотря на то, что эти файлы виртуальные — их можно просмотреть любым текстовым редактором или с помощью команд `more`, `less` или `cat`. При попытке открытия виртуального файла текстовым редактором — этот файл создаётся на лету на основе информации, содержащейся в ядре.

Рассмотрим вывод наиболее интересных файлов в этой файловой системе.

Файл, который отображает сведения об имеющихся в системе процессорах:

```
$ cat /proc/cpuinfo
processor    : 0
vendor_id    : GenuineIntel
cpu family   : 6
model        : 42
model name   : Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz
stepping     : 7
```

```
cpu MHz      : 1600.000
cache size   : 6144 KB
...
```

Отображение средней загрузки системы, количества запущенных процессов и ID последнего процесса:

```
$ cat /proc/loadavg
0.00 0.05 0.09 2/312 29599
```

Сведения о состоянии памяти:

```
# cat /proc/meminfo
MemTotal:       483488 kB
MemFree:          9348 kB
Buffers:          6796 kB
Cached:         168292 kB
```

Каталог `/proc/acpi` содержит сведения о шине ACPI, например, состояние батареи или питания.

Файл `/proc/cmdline` содержит параметры, переданные ядру при загрузке.

Файл `/proc/stat` содержит статистические данные о последней загрузке.

Файл `/proc/uptime` — содержит два числа — продолжительность работы системы и продолжительность простоя системы в секундах.

Пример отображения сведений об устройствах:

```
$ cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
...
Block devices:
  1 ramdisk
  2 fd
259 blkext
  7 loop
  8 sd
  9 md
...
```

Пример отображения сведений о поддерживаемых ядром файловых системах:

```
$ cat /proc/filesystems
nodev sysfs
nodev rootfs
nodev bdev
nodev proc
nodev cgroup
nodev cpuset
...
```

Пример отображения монтированных файловых систем:

```
$ cat /proc/mounts
rootfs / rootfs rw 0 0
none /sys sysfs rw,nosuid,nodev,noexec,relatime 0 0
none /proc proc rw,nosuid,nodev,noexec,relatime 0 0
none /dev devtmpfs
rw,relatime,size=1541192k,nr_inodes=211507,mode=755 0
0
...
```

Каталог `/proc/net` содержит данные о сети.

В каталоге `proc` содержатся подкаталоги с числовыми именами — это процессы. Каждому процессу соответствует свой каталог в файловой системе `proc`.

Таким образом, файловая система `/proc` является местом, где сосредоточены данные о системе. Для наглядности представления данных из этой файловой системы используются специальные утилиты. Утилиты сбора статистики и отслеживания состояния системы в реальном времени используют каталог `/proc` для получения актуальных данных. При необходимости администратор разрабатывает собственные сценарии для сбора статистических данных из `/proc`.

Подробная справка о файловой системе `/proc`:

```
$ man proc
```

9.1.2. Утилиты для сбора статистики

В данном разделе приведено описание утилит, которые можно использовать для мониторинга системы.

9.1.2.1. top

top — системный монитор процессов. Динамически выдаёт в режиме реального времени информации о работающей системе. По умолчанию она выдает сведения о задачах, наиболее загружающих процессор сервера, и обновляет список каждые 2 секунды.

Программа отображает сведения о процессах в виде таблицы:

– PID — идентификатор процесса;

– USER — пользователь, от которого запущен процесс;

– PR — текущий приоритет процесса;

– NI — приоритет процесса от «-20» (наивысший) до «19» (минимальный);

– VIRT — количество виртуальной памяти, выделенной процессу;

– RES — текущее использование оперативной памяти;

– SHR — память, которую процесс делит с другими процессами;

– S — текущее состояние (START, RUN, SLEEP, STOP, ZOMB, WAIT или LOCK);

– TIME — время использования процессора в секундах;

– %CPU — процент доступного времени процессора, которое использовала запущенная программа;

– %MEM — процент использования памяти;

– COMMAND — команда, запустившая процесс.

Чтобы выйти из программы top, нужно нажать клавишу [q].

Полезные интерактивные команды, которые используются при взаимодействии оператора с top:

– [Пробел] — обновить содержимое экрана;

– [h] — вывести справку о программе;

– [k] — уничтожить процесс, программа запрашивает «код процесса» и «сигнал», который будет ему послан;

– [n] — изменить количество отображаемых процессов, появляется запрос «новое количество отображаемых процессов»;

– [u] — сортировать имена процессов по имени пользователя;

– [M] — сортировать имена процессов по объему используемой памяти;

– [P] — сортировать имена процессов по загрузке процессора.

Для ввода клавиши в верхнем регистре используйте клавишу [Shift].

9.1.2.2. vmstat

Команда vmstat отображает статистические данные в реальном времени о виртуальной памяти системы.

Пример использования:

```
$ vmstat 1 10
```

Команда vmstat снимет показатели 10 раз с интервалом в одну секунду.

9.1.2.3. slabtop

Команда slabtop выводит на экран в похожем на команду top виде сведения о состоянии внутренних буферов, кэша и структур Linux ядра, доступных через /proc/slabinfo.

Команда vmstat -m отображает аналогичную информацию.

9.1.2.4. ps

Команда ps выводит на экран список наименований процессов. Без параметров ps выводит процессы, которые были запущены в течение текущей сессии, за исключением демонов.

Формат команды:

```
ps [<параметры>]
```

Параметры:

– -a или -e — показать все наименования процессов;

– -f — полный листинг описания процессов;

– -w — вывести описания процессов в несколько строк, если если размер строки превышает размер экрана.

9.1.2.5. free

Команда free отображает статистику использования оперативной памяти системы, включая кэш и буферы.

Параметры команды free:

– -b — отображать данные в байтах;

– -k — отображать данные в килобайтах;

– -m — отображать данные в мегабайтах;

– -s <секунды> — выводить статистику постоянно, обновляя данные раз в <секунду>.

Пример использования:

```
$ free
             total       used       free     shared    buffers
cached
Mem:       8181592    3994260    4187332          0      55728
3561716
-/+ buffers/cache:     376816    7804776
Swap:      4088504          0    4088504
```

9.1.2.6. w

Команда w отображает список пользователей, работающих в системе.

вывод команды w содержит следующие колонки:

– имя пользователя;

– номер tty;

– адрес, с которого произошло подключение;

– время подключения;

– время бездействия;

– время, затраченное всеми процессами в данном сеансе (JCPU);

– время, потраченное текущим процессом (PCPU);

– команда, выполняемая пользователем.

Формат команды:

```
w [<параметры>]
```

Параметры:

– -h — игнорировать информацию заголовка;

– -u — отображать текущую загрузку;

– -s — Удалить из вывода JCPU, PCPU, и время подключения.

9.1.2.7. who

Команда who используется для получения списка пользователей, работающих в системе. В выводе находятся следующие колонки: имя пользователя, номер tty, дата и время, адрес подключения.

Формат команды:

`who [<параметры>]`

Параметры:

– `-a`, `--all` — равносильно `-b -d --login -p -r -t -T -u`;

– `-b`, `--boot` — время последней загрузки системы;

– `-d`, `--dead` — выводит спящие процессы;

– `-H`, `--heading` — выводит строку заголовка (шапку) для столбцов таблицы листинга;

– `-i`, `--idle` — в листинге, после времени входа в систему, указывает количество часов и минут бездействия (простоя) пользователя в его терминале. Точка «.» в этом поле означает, что пользователь был активен в течение последней минуты, а слово `old` (старый) — бездействовал более суток (параметр `-u`);

– `-l`, `--login` — выводит системные процессы, ожидающие регистрации пользователей (т. е. свободные терминалы);

– `--lookup` — пытается узаконить (канонизировать) имена узлов, найденные через DNS;

– `-m` — выводит информацию о пользователе, связанном с текущим терминалом (о себе). Синоним команды `whoami`;

– `-p`, `--process` — выводит активные процессы, порожденные программой `init`;

– `-q`, `--count` — выводит только имена и количество зарегистрировавшихся пользователей;

– `-r`, `--runlevel` — выводит информацию о текущем уровне выполнения (`runlevel`) для пользователя в системе;

– `-s`, `--short` — выводит только имя, терминал, время регистрации, имя дистанционного (удалённого) узла (по умолчанию);

– `-t`, `--time` — если есть, то выводит информацию о последнем изменении времени в системе;

– `-T`, `-w`, `--mesg` — после каждого имени печатает символ, указывающий доступен ли для записи терминал пользователя или нет:

  – `+` — разрешается писать сообщения;

  – `-` — запрещается писать сообщения;

  – `?` — не может обнаружить терминальное устройство;

– `-u`, `--users` — то же, что и параметр `-i`;

– --message — то же, что и параметр -T;

– --writable — то же, что и параметр -T.

9.1.2.8. pstree

Команда pstree показывает запущенные процессы в виде дерева.

Формат команды:

pstree [<параметры>] [<идентификатор процесса>|<имя пользователя>]

Дерево образует корень или от указанного <идентификатор процесса> или от процесса init если идентификатор процесса не указан. Если определено имя пользователя, то дерево образует корень от процессов , принадлежащих указанному пользователю.

Параметры:

– -a — показывает команды с аргументами в командной строке;

– -c — запрещает компактную форму вывода одинаковых поддеревьев. По умолчанию, одинаковые поддеревья объединяются всякий раз когда возможно;

– -h — подсвечивает текущий процесс и его предков;

– -l — показывает длинные строки. По умолчанию, строки обрезаются до ширины дисплея или до 132 если вывод посылается на не-tty устройство или если ширина дисплея неизвестна;

– -n — сортирует процессы с одинаковым предком по идентификатору процесса вместо сортировки по имени;

– -p — показывает идентификаторы процессов. Идентификаторы процессов показываются числом, заключённым в круглые скобки после каждого имени процесса. Параметр -p запрещает вывод в компактной форме;

– -u — показывает пользователя — владельца процесса. Всякий раз когда владелец процесса отличается от владельца родителя, новый пользователь показывается после имени процесса, заключённым в круглые скобки.

9.1.2.9. uptime

Команда uptime выводит информацию о времени работы компьютера.

Формат команды:

uptime [<файл>]

Выдаёт текущее время, время работы компьютера с его включения, количество пользователей в системе и среднее количество заданий в очередях за последние 1, 5 и 15 минут. Если <файл> не указан, то используется /var/run/utmp. /var/log/wtmp используется, как <файл> в общем случае.

9.1.2.10. lsusb

Утилита lsusb отображает информацию о USB-устройствах.

Формат команды:

```
lsusb [-v]
```

Параметр -v предоставляет подробный вывод.

9.1.2.11. lspci

Утилита lspci отображает информацию о шине PCI и устройствах, которые к ней подключены.

Формат команды:

```
lspci [-v|-vv]
```

Параметр -v предоставляет подробный вывод, -vv — ещё более подробный вывод.

9.1.2.12. lscpu

Утилита lscpu отображает информацию об имеющихся в системе процессорах: количество процессоров, архитектура, производитель, семейство, модель, кэш и т. п.

Формат команды:

```
lscpu
```

9.1.3. Мониторинг файловых систем

Система предоставляет некоторые команды для мониторинга за файловыми системами, которые перечислены в таблице 14.

Таблица 14 – Перечень команд для мониторинга за файловыми системами

Команда	Краткое описание	Раздел
badblocks	Анализ диска	7.4.8

Команда	Краткое описание	Раздел
`blkid`	Получение уникального идентификатора дискового раздела	7.4.5
`df`	Проверка свободного места на дисках	7.4.12
`du`	Вычисление объёма дискового пространства, занимаемого файлами	7.4.14
`findmnt`	Поиск примонтированных файловых систем	7.4.13
`fsck`	Проверка состояния файловых систем	7.4.11
`fuser`	Идентификация использования ресурсов процессами	8.1.10
`lsof`	Информация об использовании процессами файлов	8.1.11
`lsblk`	Список блочных устройства	7.4.6
`lsattr`	Просмотр расширенных атрибутов	7.3.2
`mount`	Монтирует файловые системы, просмотр точек монтирования	7.4.2

10. АДМИНИСТРИРОВАНИЕ ЯДРА ОПЕРАЦИОННОЙ СИСТЕМЫ

Операционная система «ОСь» в качестве ядра операционной системы использует ядро Linux. Администрирование ядра позволяет настроить производительность операционной системы, загружать модули ядра и драйверы. Все операции по настройке ядра операционной системы должны выполняться квалифицированным специалистом.

10.1. Утилиты настройки ядра

10.1.1. `sysctl`

Команда `sysctl` используется для изменения параметров ядра во время выполнения. Доступные параметры перечислены в `/proc/sys/`. Команда `sysctl` может использоваться как для чтения, так и для записи параметров ядра.

Конфигурационный файл `/etc/sysctl.conf` содержит параметры ядра, устанавливаемые при загрузке операционной системы.

Формат команды:

```
sysctl [<параметры>] <переменная>[=<значение>]
```

Параметры:

`<переменная>` Имя ключа, из которого производится чтение, к примеру `kernel.ostype`. Наряду с разделителем «.» поддерживается «/».

`<переменная>=<значение>` Для того чтобы изменить значение ключа, используйте форму `<переменная>=<значение>`, где `<переменная>` — изменяемый ключ, а `<значение>` — значение, ему присваиваемое. Если значение содержит кавычки или символы, обрабатываемые оболочкой, может потребоваться заключить значение в двойные кавычки. Эта форма требует использования ключа `-w`.

`-n` Не выводить имена ключей при выводе их значений.

`-e` Игнорировать ошибки, связанные с неизвестными ключами.

`-N` Выводить только имена. Это может быть полезно для оболочек, поддерживающих настраиваемое автодополнение.

`-q` Не выводить устанавливаемые значения на стандартный вывод.

`-v` Выводить устанавливаемые значения на стандартный вывод.

`-w` Используется для изменения значений ключей.

`-p` Загрузить настройки из указанного файла или `/etc/sysctl.conf`, если файл не указан.

`-a` Вывести все доступные в данный момент настройки.

`-A` Вывести все доступные в данный момент настройки в табличном виде.

10.1.2. `modprobe`

Команда `modprobe` предназначена для загрузки/выгрузки модулей ядра. Модули загружаются с учётом зависимостей.

Формат команды:

```
modprobe [<параметры>] [<имя модуля>] [<параметры модуля...>]
```

Параметры:

`-v, --verbose` Выводить подробные сообщения о том, что делает программа.

`-C, --config` Заменяет конфигурационный файл по умолчанию.

`-c, --showconfig` Вывести файл конфигурации и завершить работу.

`-n, --dry-run` При вставке или удалении модулей (или запуская команды установки или удаления) на самом деле ничего не делает, полезна для диагностики.

`-i, --ignore-install, --ignore-remove` Игнорировать в файле конфигурации команды `install` или `remove`.

`-q, --quiet` «Тихий» режим — не выводить сообщения об ошибках.

`-r, --remove` Заставит `modprobe` удалить, а не вставить модуль.

`-w, --wait` Заблокировать модуль в ядре (внутри ядра модуль обрабатывает код самостоятельно), ожидая когда счётчик ссылок на указанный модуль достигнет нуля.

`-f, --force` Принудительно загрузить модуль, несмотря на ошибки (опасная операция).

`-l, --list` Список всех модулей, совпадающих с заданным шаблоном.

`-s, --syslog` Заставляет выводить любые сообщения об ошибках через механизм `syslog`.

`--show-depends` Список зависимостей модуля (или псевдонима), включая сам модуль.

`-o, --name` Пытается переименовать модуль, вставленный в ядро.

`--first-time` Завершиться с ошибкой, если модуль уже загружен.

`--dump-modversions` Вывести список с информацией о версиях модулей, требуемых указанным модулем.

`--use-blacklist` Применить совпадение модуля с записью в чёрном списке к запрошенному имени, а не только к запрошенному псевдониму.

`-V, --version` Показать версию программы и завершить работу.

10.1.3. `modinfo`

`modinfo` — программа для просмотра информации о модуле ядра.

Формат команды:

```
modinfo [-F <поле>] [<файл...>|<имя модуля...>]
```

По умолчанию `modinfo` выводит каждый атрибут модуля для наглядности в виде `имя_поля:значение`. Имя файла выводится таким же образом. Можно указать конкретное поле для вывода параметром `-F`.

10.1.4. `insmod`

`insmod` — простая программа для вставки модуля в ядро. Является сильно упрощённым аналогом `modprobe`.

Формат команды:

```
insmod [<файл с модулем>] [<параметры модуля>]
```

10.1.5. `lsmod`

`lsmod` — программа для просмотра состояния модулей в ядре, показывая какие модули ядра в настоящее время загружены.

Формат команды:

```
lsmod
```

10.1.6. `rmmod`

`rmmod` — простая программа для удаления модуля из ядра. Она не учитывает зависимости, в отличие от `modprobe`.

Формат команды:

```
rmmod [<параметры>] [<имя модуля>]
```

Параметры:

`-v, --verbose` Выводить подробные сообщения о том, что делает программа.

`-f, --force` Удалить модули, которые используются или не спроектированы удаляемыми, или были помечены, как небезопасные (очень опасно для стабильности системы).

`-w, --wait` Изолирует модуль, и ждёт когда он перестанет использоваться.

`-s, --syslog` Отправлять сообщения об ошибках в `/var/log/messages`, а не на стандартный поток диагностики.

`-V, --version` Показать версию программы и завершить работу.

10.1.7. `dmesg`

`dmesg` используется для просмотра и настройки уровня выходных сообщений, которые появляются в процессе начальной загрузки ядра и хранятся в кольцевом буфере. Это полезная команда может помочь пользователю увидеть, как ядро распознавало устройства во время начальной загрузки и получить диагностическую информацию, которую выводили драйвера. Эту информацию можно использовать для устранения проблем в работе некоторых устройств.

Формат команды:

```
dmesg [-c] [-n <уровень>] [-s <размер буфера>]
```

Параметры:

`-c` Очистить содержимое буфера после вывода.

`-s <размер буфера>` Использовать указанный размер буфера.

`-n <уровень>` Установить уровень, на котором сообщения системного журнала будут выводиться на консоль. Например, `-n 1` предотвращает вывод всех сообщений, за исключением явно тревожных. Несмотря на это, сообщения всех уровней будут записываться в `/proc/kmsg`, таким образом, при помощи службы `syslogd` можно будет контролировать источник появления сообщения ядра. Когда используется параметр `-n`, `dmesg` не будет выводить или очищать кольцевой буфер сообщений ядра. При использовании обоих параметров, только последний параметр командной строки будет иметь эффект.

10.2. Отключение питания и перезагрузка

10.2.1. `poweroff`

Команда `poweroff` выполняет выключение питания компьютера.

Формат команды:

```
poweroff
```

10.2.2. `reboot`

Команда `reboot` выполняет перезагрузку компьютера.

Формат команды:

```
reboot
```

10.3. Управление параметрами загрузки ядра

Старт операционной системы начинается с загрузочного меню. При помощи загрузочного меню, ядру можно передать различные параметры до начала загрузки операционной системы (см. раздел «Загрузочное меню»).

11. АДМИНИСТРИРОВАНИЕ СЕТИ

11.1. Диагностика сети

Команда `ifconfig` выводит информацию об имеющихся в системе сетевых интерфейсах, их состоянии и адресах. Команда требует привилегий системного администратора.

Примеры:

```
# ifconfig
(...)
eth0      Link encap:Ethernet  HWaddr 00:50:BA:8F:61:7A
          inet addr:192.168.0.2  Bcast:192.168.0.255
Mask:255.255.255.0
          inet6 addr: fe80::50:ba8f:617a/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1498792 errors:0 dropped:0
overruns:0 frame:0
          TX packets:1284980 errors:0 dropped:0
overruns:0 carrier:0
          collisions:1984 txqueuelen:100
          RX bytes:485691215 (463.1 Mb)  TX
 bytes:123951388 (118.2 Mb)
          Interrupt:11 Base address:0xe800
```

Просмотр только сведений о конкретном интерфейсе:

```
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:50:BA:8F:61:7A
          inet addr:192.168.0.2  Bcast:192.168.0.255
Mask:255.255.255.0
          inet6 addr: fe80::50:ba8f:617a/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1498792 errors:0 dropped:0
overruns:0 frame:0
          TX packets:1284980 errors:0 dropped:0
overruns:0 carrier:0
          collisions:1984 txqueuelen:100
```

```
          RX bytes:485691215 (463.1 Mb)  TX
bytes:123951388 (118.2 Mb)
          Interrupt:11 Base address:0xe800
```

11.2. Проверка доступности сети

Проверка доступности узла в сети или наличия доступа в интернет выполняется при помощи команды `ping`, например:

```
$ ping www.ru
PING www.ru (194.87.0.50) 56(84) bytes of data.
64 bytes from www.ru (194.87.0.50): icmp_req=1 ttl=54
time=5.94 ms
64 bytes from www.ru (194.87.0.50): icmp_req=2 ttl=54
time=5.29 ms
64 bytes from www.ru (194.87.0.50): icmp_req=3 ttl=54
time=5.26 ms
64 bytes from www.ru (194.87.0.50): icmp_req=4 ttl=54
time=4.85 ms
^C~--- www.ru ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time
3003ms
rtt min/avg/max/mdev = 4.856/5.339/5.945/0.396 ms
```

Для окончания работы программы нажмите сочетание клавиш [Ctrl+C].

Для ограничения числа пакетов, которые программа должна отослать по заданному адресу, нужно указать параметр `-c` и количество пакетов, например, отправка четырёх ICMP пакетов на сервер `www.ru`:

```
$ ping -c 4 www.ru
```

Для трассировки маршрута используется команда `traceroute`. Данная утилита последовательно опрашивает все шлюзы, через которые проходят пакеты до указанного адреса, например:

```
$ traceroute www.ru
traceroute to www.ru (194.87.0.50), 30 hops max, 60 byte
packets
 1  192.168.10.1 (192.168.10.1)  10.898 ms  10.885 ms
10.867 ms
```

```
...
 8  195.239.134.154 (195.239.134.154)  20.747 ms  21.806 ms
Ruscomnet-gw.Moscow.gldn.net (194.186.0.38)  22.985 ms
 9  DEMOS-gw.ruscomnet.ru (80.249.128.166)  25.399 ms27.007
ms  27.669    ms
10  iki-c1-vl10.demos.net (194.87.0.111)  40.944 ms41.642 ms
 42.633    ms
11  kw1.demos.ru (194.87.5.116)  30.541 ms31.241 ms  10.438
ms
```

11.3. Настройка имени хоста

Для того чтобы изменить имя хоста, воспользуйтесь командой `z-adm hostname <новое имя хоста>`.

Не забудьте отредактировать `/etc/hosts`, если там было записано старое имя хоста.

При возникновении проблем выполните перезагрузку сети (`service network restart`) или компьютера (`reboot`).

11.4. Базовые параметры сети

При настройке TCP/IP сети используются следующие основные параметры.

IP-адрес — уникальный адрес машины, записываемый в формате [0-255].[0-255].[0-255].[0-255]. Для локальной сети часто используются адреса вида 192.168.1.35.

Маска подсети — записывается аналогично IP-адресу и разделяет IP-адрес на две части, первая относится к адресу подсети, вторая — к адресу узла. Например адрес 192.168.1.2 с маской 255.255.255.0 принадлежит подсети 192.168.1, а часть адреса узла — 2.

IP адрес шлюза — это адрес машины, являющейся шлюзом для связи с более крупной подсетью. Шлюзов может быть несколько, если компьютер подключен к нескольким сетям одновременно. Адрес шлюза не используется в изолированных сетях (не подключенных к глобальной сети).

IP-адрес сервера имен (DNS — сервера) — адрес сервера преобразующего имена хостов в IP адреса. Как оправило, это адрес кэширующего DNS-сервера, обменивающимся информацией с другим DNS-сервером более высокого уровня.

11.5. Настройка IP-адресов

Настройка IP-адресов сетевых интерфейсов осуществляется через файлы `/etc/sysconfig/network-scripts/ifcfg-<имя интерфейса>`.

Пример параметров для динамического IP:

```
DEVICE=eth0
BOOTPROTO=dhcp
ONBOOT=yes
```

Пример статического IP:

```
DEVICE=eth0
BOOTPROTO=static
ONBOOT=yes
IPADDR=192.168.1.101
NETMASK=255.255.255.0
GATEWAY=192.168.1.1
DNS1=192.168.1.2
DNS2=192.168.1.3
USERCTL=no
```

11.6. Настройка сетевых интерфейсов

Сетевой интерфейс Ethernet обычно имеет имя вида `ethX` или `enp*`. Параметры указываются в файле `/etc/sysconfig/network-scripts/ifcfg-<имя интерфейса>`.

Основные параметры:

– BOOTPROTO=<protocol> — протокол загрузки (none — не использовать, dhcp — использовать DHCP, bootp — использовать BOOTP);

– DEVICE=<name> — имя интерфейса физического устройства;

– IPADDR=<address> — IP-адрес;

– GATEWAY=<address> — адрес шлюза;

– NETMASK=<mask> — маска сети;

– ONBOOT=<answer> — автоматически поднимать интерфейс при старте системы: yes (да) или no (нет);

– DNS{1,2}=<address> — адреса DNS-серверов, которые будут записаны в /etc/resolv.conf;

– USERCTL=<answer> — пользователь может управлять соединением: yes (да) или no (нет).

Дополнительные параметры:

– DHCP_HOSTNAME — данный параметр нужен только в том случае, когда DHCP-сервер выдаёт адрес только после отправки имени хоста;

– ETHTOOL_OPTS=<options> — специфичные для устройства параметры, поддерживаемые утилитой ethtool, например: ETHTOOL_OPTS=”autoneg off speed 100 duplex full”;

– HWADDR=<MAC-address> — MAC-адрес физического устройства, полезно для машин с несколькими сетевыми картами, чтобы гарантировать привязку номера интерфейса к определённому физическому устройству;

– MACADDR=<MAC-address> — MAC-адрес, который будет использоваться вместо MAC-адреса физического устройства, нельзя использовать совместно с HWADDR;

– PEERDNS=<answer> — модифицировать файл /etc/resolv.conf: yes (да) или no (нет), при использовании DHCP yes — по умолчанию;

– MASTER=<bond-interface> — при настройке бондинга — имя связанного с данным устройством бондинг-интерфейса;

– SLAVE=<answer> — используется при настройке бондинга совместно с директивой MASTER: yes (устройство контролируется другим интерфейсом, указанным в MASTER) или no (не контролируется);

– SRCADDR=<address> — адрес-источник для исходящих пакетов.

11.7. Псевдонимы сетевых интерфейсов

Псевдонимы (или алиасы) сетевых интерфейсов позволяют подключаться одновременно к нескольким подсетям, используя при этом одно и то же физическое сетевое устройство.

Для создания псевдонима, достаточно создать файл с именем `ifcfg-<интерфейс>:N`, где `<интерфейс>` — обозначение существующего интерфейса, `N` — номер псевдонима. В этом файле нужно указать те же параметры, как и у обычного физического интерфейса (`IPADDR`, `NETMASK`, `GATEWAY`, `DNS1`, `DNS2` и т. п.), а также добавить параметр `REALDEVICE=<интерфейс>`.

Можно создать сразу несколько псевдонимов с серией IP-адресов. Для этого создайте файл с именем ifcfg-<интерфейс>-range, в котором нужно указать примерно следующие параметры:

```
IPADDR_START=192.168.1.105
IPADDR_END=192.168.1.115
CLONENUM_START=5
```

Данная конфигурация создаст 10 псевдонимов с адресами от 192.168.1.105 до 192.168.1.115, причём нумерация псевдонимов начнётся с :5.

11.8. Управление сетевыми интерфейсами

Для того, чтобы отключить («опустить») или включить («поднять») интерфейс, необходимо использовать команду ifconfig:

```
# ifconfig <интерфейс> up|down
```

Для запуска, остановки, перезапуска всех интерфейсов системы необходимо использовать службу networking:

```
# service networking start|stop|restart
```

11.9. Изменение сетевых настроек

Команда ifconfig используется для настройки сетевых интерфейсов. Команда должна использоваться при загрузке системы для настройки адресов каждого сетевого интерфейса и после загрузки для изменения параметров сетевых интерфейсов. Если команда введена без аргументов, ifconfig выдает сведения о состоянии активных интерфейсов. Если в качестве аргумента указан какой-либо интерфейс, то выдается информация только о состоянии этого интерфейса; если указан один аргумент -a, выдается информация о состоянии всех интерфейсов, даже отключенных. Иначе команда конфигурирует указанный интерфейс.

Производить настройку сети должен системный администратор.

Команда ifconfig имеет следующий синтаксис:

```
ifconfig [-v] [-a] [-s] [<интерфейс>]
ifconfig [-v] <интерфейс> [aftype] <опции> | <адрес> ...
```

Опции команды ifconfig:

– <интерфейс> — имя интерфейса (например, eth0);

– up — вызывает активизацию интерфейса, задается неявно при присвоении адреса интерфейсу;

– down — вызывает остановку работы драйвера для интерфейса;

– [-]arp — включает или отключает использование протокола ARP для интерфейса;

– [-]promisc — включает или отключает неразборчивый режим (promiscuous mode) работы интерфейса, в этом режиме все проходящие по сети пакеты принимаются интерфейсом;

– [-]allmulti — включает или отключает режим all-multicast, в этом режиме все многоадресные (multicast) пакеты в сети принимаются интерфейсом;

– metric N — устанавливает метрику интерфейса;

– mtu N — устанавливает максимальный размер пакета (Maximum Transfer Unit — MTU) для интерфейса;

– адрес — IP-адрес, присваиваемый интерфейсу;

– netmask адрес — устанавливает маску сети IP для этого интерфейса, по умолчанию используется обычная маска сети класса A, B или C (что определяется по IP-адресу интерфейса), но устанавливается необходимое значение;

– add адрес/длина_префикса — добавляет адрес IPv6 для интерфейса;

– del адрес/длина_префикса — удаляет адрес IPv6 для интерфейса;

– irq адрес — устанавливает аппаратное прерывание, используемое устройством, не для всех устройств можно динамически менять значение IRQ;

– media тип — устанавливает физический порт или тип носителя, используемый устройством, не для всех устройств допустимо изменение параметра, и для разных устройств могут поддерживаться различные значения, специальный тип носителя auto используется для автоматического определения типа носителя;

– [-]broadcast <адрес> — аргумент адрес задает соответствующий протоколу широковещательный адрес для интерфейса, в противном случае устанавливает (или сбрасывает) флаг IFF_BROADCAST для интерфейса.

Пример. Задать основной IP-адрес и маску сети для интерфейса eth0:

```
# ifconfig eth0 192.168.0.1 netmask 255.255.255.0
```

Присвоение дополнительного IP-адреса интерфейсу eth0:

```
# ifconfig eth0:0 10.10.0.1 netmask 255.255.255.0
```

11.10. Изменение DNS и создание доменных имен

Для добавления или изменения DNS-серверов, открыть для редактирования файл `/etc/resolv.conf` и написать в нем:

```
nameserver <1-й DNS сервер>
nameserver <2-й DNS сервер>
```

Для создания доменных имен отредактировать файл `/etc/hosts`.

Чтобы временно поменять MAC-адрес сетевой карты с интерфейсом `eth0`, нужно использовать команду:

```
# ifconfig eth0 hw ether 00:01:02:03:04:05
```

11.11. Маршрутизация

Маска подсети позволяет определить все узлы, находящиеся в той же локальной сети. Пакеты к ним будут доставляться напрямую через канальный уровень.

Таблицу, управляющую маршрутизацией пакетов, просматривают с помощью команды `netstat -r` или `route`, обе команды имеют ключ `-n` для выдачи IP-адреса, а не имени компьютера.

Пример выполнения команды `route`:

```
# route -n
Kernel IP routing table
Destination     Gateway         Genmask         FlagsMetric
Ref    Use    Iface
192.168.1.0     0.0.0.0         255.255.255.0   U     00
   0    eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     00
   0    lo
0.0.0.0         192.168.1.1     0.0.0.0         UG    00
   0    eth0
```

Компьютер или аппаратное устройство, осуществляющее маршрутизацию между локальной сетью и Internet, называется шлюзом.

Параметры маршрутизации задаются с помощью команды `route`, например:

```
# route -f <операция> -тип <адресат> <шлюз> <интерфейс>
```

Аргумент `<операция>` принимает одно из двух значений: `add` (добавить маршрут) или `delete` (удалить маршрут). Аргумент `<адресат>` может быть IP-

адресом машины, IP-адресом сети или ключевым словом `default`. Аргумент `<шлюз>` — это IP-адрес компьютера, на который следует пересылать пакет, этот компьютер должен иметь прямую связь с компьютером-получателем сообщения, например:

```
# route -f
```

удаляет из таблицы данные обо всех шлюзах.

Необязательный аргумент `<тип>` принимает значения `net` или `host`. В первом случае в поле адресата указывается адрес сети, а во втором — адрес конкретного компьютера (хоста).

Настройка маршрутизации выполняется по трем интерфейсам:

– локальный интерфейс (`lo`);

– интерфейс для платы «Ethernet» (`eth0`);

– интерфейс для последовательного порта (`PPP` или `SLIP`).

Локальный интерфейс поддерживает сеть с IP-номером `127.0.0.1`.

Поэтому для маршрутизации пакетов с адресом `127....` используется команда:

```
# route add -net 127.0.0.1 lo
```

Если для связи с локальной сетью используется одна плата Ethernet, и все машины находятся в этой сети (сетевая маска `255.255.255.0`), то для настройки маршрутизации достаточно вызвать:

```
# route add -net 192.168.36.0 netmask 255.255.255.0 eth0
```

Если имеется насколько интерфейсов, то необходимо определиться с сетевой маской и вызвать команду `route` для каждого интерфейса. Так как маршрутом может быть очень много, обычно задают маршрут по умолчанию, который используется для отправки всех пакетов, не указанных явно в таблице маршрутизации. Маршрут по умолчанию настраивается следующей командой:

```
# route add default gw 192.168.1.1 eth0
```

Параметр `gw` указывает программе `route`, что следующий аргумент — это IP-адрес или имя маршрутизатора, на который надо отправлять все пакеты, соответствующие этой строке таблицы маршрутизации.

11.12. Настройка даты и времени

Утилита настройки даты и времени позволяет установить компьютерные часы.

11.12.1. Ручная настройка даты и времени

Наберите команду `date`, чтобы посмотреть текущие дату и время:

```
$ date
Ср. нояб.  9 10:01:23 MSK 2011
```

Если необходимо изменить их, воспользуйтесь следующим форматом команды:

```
# date MMDDhhmm[[CC]YY][.ss]
```

`MM` — месяц, `DD` — день месяца, `hh` — часы, `mm` — минуты, `CCYY` — 4 цифры года (`CC` — первые две цифры года, `YY` — последние две цифры года), `ss` — секунды.

Если опущены первые две цифры года, то если указаны годы 00 до 69 — автоматически подразумеваются годы 2000–2069, если указаны годы от 70 до 99 — подразумеваются годы 1970–1999.

Пример команды:

```
# date 110615352011.30
```

Эта команда изменит текущие дату и время.

11.13. Межсетевой экран

Межсетевой экран осуществляет контроль сетевого трафика, проходящего через данный компьютер. Фильтрацию пакетов выполняет фильтр пакетов `iptables`. Данный фильтр позволяет выполнять следующие задачи:

– фильтрацию пакетов (`netfilter`) — это механизм, который, основываясь на некоторых правилах, разрешает или запрещает передачу информации, проходящей через него, с целью ограждения некоторой подсети от внешнего доступа, или, наоборот, для недопущения выхода наружу. Фильтр пакетов может определять правомерность передачи информации на основе только заголовков IP-пакетов, а может анализировать и их содержимое, т. е. использовать данные протоколов более высокого уровня;

– трансляцию сетевых адресов — это подмена некоторых параметров в заголовках IP-пакетов. Используется для сокрытия реальных IP-адресов компьютеров защищаемой ЛВС, а также для организации доступа из ЛВС с компьютерами, не имеющими реальных IP-адресов, к глобальной сети;

– прозрачное проксирование — это переадресация пакетов на другой порт компьютера. Обычно используется для того, чтобы заставить пользователей из ЛВС пользоваться прокси-сервером маршрутизатора без дополнительного конфигуриро-

вания их клиентских программ. Настройка рассмотренных механизмов (фильтрация пакетов, трансляция сетевых адресов и прозрачное проксирование) выполняется командой `iptables`.

11.13.1. Настройка межсетевого экрана

Настройка рассмотренных механизмов (фильтрация пакетов, трансляция сетевых адресов и прозрачное проксирование) выполняется командой `iptables`.

Каждое правило — это строка, содержащая в себе критерии, определяющие, подпадает ли пакет под заданное правило, и действие, которое необходимо выполнить в случае выполнения критерия. Правила записываются следующим образом:

```
iptables [-t table] command [match] [target/jump]
```

Если в правило не включается спецификатор `[-t table]`, то по умолчанию предполагается использование таблицы `filter`, если же предполагается использование другой таблицы, то это требуется указать явно. Спецификатор таблицы также можно указывать в любом месте строки правила, однако более или менее стандартным считается указание таблицы в начале правила. Далее, непосредственно за именем таблицы, должна стоять команда. Если спецификатора таблицы нет, то команда всегда должна стоять первой. Команда определяет действие `iptables`, например вставить, добавить в конец цепочки или удалить правило и т. п.

Раздел `matches` задает критерии проверки, по которым определяется, подпадает ли пакет под действие этого правила или нет. Здесь можно указать самые разные критерии — и IP-адрес источника пакета или сети, и сетевой интерфейс, и т. д.

Раздел `target` указывает, какое действие должно быть выполнено при условии выполнения критериев в правиле. Здесь можно заставить ядро передать пакет в другую цепочку правил, «сбросить» пакет и забыть про него, выдать на источник сообщение об ошибке и т. п.

Когда пакет приходит на сетевой фильтр, то он сначала попадает на сетевое устройство, перехватывается соответствующим драйвером и далее передается в ядро. Затем пакет проходит несколько таблиц и после передается либо локальному приложению, либо переправляется на другой компьютер. Порядок следования пакета приводится в таблице 15.

Таблица 15 – Порядок следования пакета

Шаг	Таблица	Цепочка	Описание
1	=	=	Кабель
2	=	=	Сетевой интерфейс (например eth0)
3	mangle	PREROUTING	Используется для внесения изменений в заголовок пакета
4	nat	PREROUTING	Используется для трансляции сетевых адресов DNAT
5	=	=	Принятие решения о дальнейшей маршрутизации, т. е. в этой точке решается, куда пойдет пакет — локальному приложению или на другой узел сети
6	filter	FORWARD	Попадают только те пакеты, которые идут на другой компьютер. Вся фильтрация транзитного трафика должна выполняться здесь. Через эту цепочку проходит трафик в обоих направлениях, поэтому обязательно учитывать это обстоятельство при написании правил фильтрации
7	nat	POSTROUTING	Предназначена в первую очередь для SNAT. Не использовать для фильтрации без особой необходимости. Здесь же выполняется и маскарадинг
8	=	=	Выходной сетевой интерфейс (например, eth1)
9	=	=	Кабель

Пакет проходит несколько этапов, прежде чем он будет передан далее. На каждом из них пакет может быть остановлен. Цепочку FORWARD проходят все пакеты, которые движутся через сетевой фильтр. В таблице 16 представлен порядок движения пакета, предназначенного локальному процессу/приложению.

Таблица 16 – Порядок движения пакета для локального процесса

Шаг	Таблица	Цепочка	Описание
1	=	=	Кабель

Шаг	Таблица	Цепочка	Описание
2	=	=	Сетевой интерфейс (например eth0)
3	mangle	PREROUTING	Обычно используется для внесения изменений в заголовок пакета
4	nat	PREROUTING	Преобразование адресов DNAT. Фильтрация пакетов здесь допускается только в исключительных случаях
5	=	=	Принятие решения о дальнейшей маршрутизации, т. е. в этой точке решается, куда пойдет пакет — локальному приложению или на другой узел сети
6	filter	INPUT	Фильтрация входящего трафика. Все входящие пакеты, адресованные локальному приложению, проходят через эту цепочку, независимо от того, с какого интерфейса они поступили
7	=	=	Локальный процесс/приложение

Пакеты идут через цепочку INPUT, а не через FORWARD. В таблице 17 представлен порядок движения пакетов, созданных локальными процессами.

Таблица 17 – Порядок движения пакетов, созданных локальными процессами

Шаг	Таблица	Цепочка	Описание
1	=	=	Локальный процесс
2	mangle	OUT	Сетевой интерфейс (например eth0)
3	filter	OUT	Внесение изменений в заголовок пакета. Фильтрация, выполняемая в этой цепочке, может иметь негативные последствия
4	=	=	Принятие решения о маршрутизации. Здесь решается — куда пойдет пакет дальше
5	nat	POSTROUTING	Здесь выполняется SNAT. Не следует в этой цепочке производить фильтрацию пакетов во избежание нежелательных побочных эффектов. Однако и здесь можно останавливать пакеты, применяя политику по умолчанию — DROP

Шаг	Таблица	Цепочка	Описание
6	=	=	Сетевой интерфейс (например, `eth0`)
7	=	=	Кабель

11.13.2. Таблица `MANGLE`

Данная таблица предназначена для внесения изменений в заголовки пакетов, то есть в этой таблице можно устанавливать биты `TOS` и т. д. В этой таблице не следует производить любого рода фильтрацию, маскировку или преобразование адресов (`DNAT`, `SNAT`).

В этой таблице допускается выполнять только те действия, которые перечислены в таблице 18.

Таблица 18 – Допустимые действия `MANGLE`

Действие	Описание
`TOS`	Выполняет установку битов поля `TOS` в пакете. Это поле используется для назначения сетевой политики обслуживания пакета, т. е. задает желаемый вариант маршрутизации
`TTL`	Используется для установки значения поля `TTL` пакета.
`MARK`	Устанавливает специальную метку на пакет, которая затем может быть проверена другими правилами в `iptables` или другими программами. С помощью меток можно управлять маршрутизацией пакетов, ограничивать трафик и т. п.

Таблица имеет две цепочки:

– `PREROUTING` — используется для внесения изменений на входе в сетевой фильтр перед принятием решения о маршрутизации;

– `OUTPUT` — для внесения изменений в пакеты, поступающие от приложений внутри сетевой фильтр.

Таблица `mangle` не должна использоваться для преобразования сетевых адресов или маскарадинга, поскольку для этих целей имеется таблица `nat`.

11.13.3. Таблица `NAT`

Эта таблица используется для выполнения преобразований сетевых адресов `NAT`. Только первый пакет из потока проходит через цепочки этой таблицы. Транс-

ляция адресов или маскировка применяются ко всем последующим пакетам в потоке автоматически.

Таблица имеет две цепочки:

– PREROUTING — используется для внесения изменений в пакеты на входе в сетевой фильтр;

– OUTPUT — используется для преобразования пакетов, созданных приложениями внутри сетевой фильтр, перед принятием решения о маршрутизации.

11.13.4. Таблица FILTER

В этой таблице содержатся наборы правил для выполнения фильтрации пакетов. Пакеты могут пропускаться далее либо отвергаться в зависимости от их содержимого. В таблице filter можно выполнить DROP, LOG, ACCEPT или REJECT без каких-либо сложностей, как в других таблицах. Имеется три встроенных цепочки:

– FORWARD — используется для фильтрации пакетов, идущих транзитом через сетевой фильтр;

– INPUT — проходят пакеты, которые предназначены локальным приложениям (сетевому фильтру);

– OUTPUT — используется для фильтрации исходящих пакетов, сгенерированных приложениями на самом сетевом фильтре.

11.13.5. Команды iptables

Ниже приводится список команд, которые используются в iptables, и правила их использования. Посредством команд iptables узнает, что необходимо выполнить.

-A, --append Добавляет новое правило в конец заданной цепочки. Пример iptables -A INPUT ....

-D, --delete Удаляет правило из цепочки. Команда имеет два формата записи, первый — когда задается критерий сравнения с параметром -D, второй — порядковый номер правила. Если задается критерий сравнения, то удаляется правило, которое имеет в себе этот критерий, если задается номер правила, то будет удалено правило с заданным номером. Счет правил в цепочках начинается с единицы. Пример iptables -D INPUT --dport 80 -j DROP, iptables -D INPUT 1.

-E, --rename-chain Выполняет переименование пользовательской цепочки. В примере цепочка allowed будет переименована в цепочку disallowed. Эти

переименования не изменяют порядок работы. Пример `iptables -E allowed disallowed` Команда должна быть указана всегда.

`-F, --flush` Сброс (удаление) всех правил из заданной цепочки (таблицы). Если имя цепочки и таблицы не указывается, то удаляются все правила во всех цепочках. Пример `iptables -F INPUT`.

`-R, --replace` Данная команда заменяет одно правило другим. В основном она используется во время отладки новых правил. Пример `iptables -R INPUT 1 -s 192.168.0.1 -j`.

`-I, --insert` Вставляет новое правило в цепочку. Число, следующее за именем цепочки, указывает номер правила, перед которым следует вставить новое правило. В примере выше указывается, что данное правило должно быть первым в цепочке `INPUT`. Пример: `iptables -I INPUT 1 --dport 80 -j ACCEPT`.

`-L, --list` Вывод списка правил в заданной цепочке. В нижеприведенном примере предполагается вывод правил из цепочки `INPUT`. Если имя цепочки не указывается, то выводится список правил для всех цепочек. Формат вывода зависит от наличия дополнительных ключей в команде, например `-n`, `-v` и пр. Пример `iptables -L INPUT`.

`-N, --new-chain` Создается новая цепочка с заданным именем в заданной таблице. В нижеприведенном примере создается новая цепочка с именем `allowed`. Имя цепочки должно быть уникальным и не должно совпадать с зарезервированными именами цепочек и действий (`DROP`, `REJECT` и т. п.). Пример `iptables -N allowed`.

`-P, --policy` Определяет политику по умолчанию для заданной цепочки. Политика по умолчанию определяет действие, применяемое к пакетам, не попавшим под действие ни одного из правил в цепочке. В качестве политики по умолчанию допускается использовать `DROP`, `ACCEPT` и `REJECT`. Пример `iptables -P INPUT DROP`.

`-X, --delete-chain` Удаление заданной цепочки из заданной таблицы. Удаляемая цепочка не должна иметь правил и не должно быть ссылок из других цепочек на удаляемую цепочку. Если имя цепочки не указано, то будут удалены все цепочки, определенные командой `-N` в заданной таблице. Примеры: `iptables -X allowed` или `iptables -P INPUT DROP`.

`-Z, --zero` Обнуление всех счетчиков в заданной цепочке. Если имя цепочки не указывается, то подразумеваются все цепочки. При использовании ключа `-v`

совместно с командой `-L` на вывод будут поданы и состояния счетчиков пакетов, попавших под действие каждого правила. Допускается совместное использование команд `-L` и `-Z`. В этом случае будет выдан сначала список правил со счетчиками, а затем произойдет обнуление счетчиков.

12. СЕРВЕРНЫЕ СРЕДСТВА

Операционная система «ОСь» предоставляет некоторые серверные возможности: организация удалённого доступа, файловый сервер, станция печати, серверы LDAP, DHCP, DNS, PXE.

12.1. Серверная инфраструктура

Сервер каталогов (LDAP):

– хранение информации об учётных записях пользователей и групп, авторизация в других модулях;

– общие конфигурационные параметры комплекса;

– различных вспомогательных каталоги для обслуживания прикладных систем (например: организационная структура, справочники контрагентов и т. д.), укладывающихся в иерархическую модель данных.

Файловый сервер предназначен для предоставления сервиса удаленной работы с файловой структурой по протоколу NFS.

Сервер печати CUPS предназначен для предоставления сервиса удаленной печати.

Сервер имен предназначен для разрешения доменных имен Интернет (DNS).

Серверы удаленной загрузки (PXE, BOOTP, TFTP) предназначены для осуществления возможности сетевой загрузки бездисковых рабочих станций и терминалов.

12.2. Сервер SSH

SSH — сетевой протокол прикладного уровня, позволяющий производить удалённое управление операционной системой и туннелирование TCP-соединений (например, для передачи файлов). Схож по функциональности с протоколами Telnet и Rlogin. SSH-клиенты и SSH-серверы доступны для большинства сетевых операционных систем, что позволяет подключаться к ОС с удалённых машин. SSH позволяет безопасно передавать в незащищённой среде любой другой сетевой протокол.

Возможности SSH-сервера:

– удаленное управление компьютером при помощи командной оболочки;

– сжатие передаваемых данных, что удобно, например, для удалённого запуска клиентов X Window System;

– передача графических изображений окон по сети, это позволяет работать удаленно не только в командной оболочке, но и в графической среде;

– SSH-туннель — туннель, создаваемый посредством SSH-соединения.

Подробную информацию об утилите ssh можно получить, выполнив команду `man ssh`.

12.2.1. Удалённый вход в систему по протоколу SSH

Для того, чтобы выполнить вход на удалённую машину под своим именем пользователя, выполните команду:

```
$ ssh <адрес>
```

Где `<адрес>` — IP-адрес удалённого хоста или имя хоста. При первом входе на удалённую машину, `ssh`-клиент запросит подтверждение, нужно набрать `yes` и нажать клавишу [Enter]. При каждом входе `ssh`-клиент спрашивает пароль, нужно ввести пароль пользователя удалённого хоста.

Для входа в систему под другим именем пользователя, выполните команду:

```
$ ssh <имя пользователя>@<адрес>
```

Данный пользователь должен существовать на удалённом хосте, также потребуется ввести пароль этого пользователя.

12.2.2. Копирование файлов при помощи SSH

SSH предоставляет удобный механизм переноса файлов между компьютерами без необходимости настраивать FTP/Samba/NFS сервер. Данная возможность доступна сразу после установки системы. Копировать можно как файлы по отдельности, так и каталоги целиком.

Для того, чтобы копировать файлы, используется команда `scp`. Принцип её работы схож с командой `ssh`, при операциях с удалёнными хостами нужно вводить пароль пользователя.

Формат команды:

```
scp [-r] <источник> <назначение>
```

Источником и назначением могут быть как локальные файлы, так и удалённые машины. Локальные файлы могут выступать как в качества источника (копирование с локального хоста на удалённый), так и в качестве назначения (копирование с удалённого хоста на локальный). Можно копировать файлы с одного удалённого хоста на другой.

Формат источника и назначения для удалённого хоста:

```
[<имя пользователя>@]<адрес>:<путь>
```

Здесь:

– имя пользователя — имя пользователя, от имени которого будут осуществляться файловые операции;

– адрес — адрес удалённого хоста, как в команде `ssh`;

– путь — полный путь до файла/каталога на удалённом хосте или относительный путь (относительно домашнего каталога).

Примеры.

Копирование с локального хоста на удалённый в домашний каталог пользователя с таким же именем:

```
$ scp my_file 192.168.0.2:~
```

Копирование файла с удалённого хоста на локальный из домашнего каталога пользователя `ivan`:

```
$ scp ivan@192.168.0.2:~/file .
```

Рекурсивное копирование всего домашнего каталога пользователя `ivan` в локальный каталог `/tmp`:

```
$ scp -r ivan@192.168.0.2:~ /tmp
```

12.2.3. Настройка входа для системного администратора

Для того, чтобы иметь возможность подключения к ОС с удалённых машин, используя учётную запись системного администратора, необходимо настроить систему защиты:

```
# setsebool -P ssh_sysadm_login on
```

В противном случае подключение к данной машине по протоколу SSH с учётной записью системного администратора будет невозможно.

Для того, чтобы отключить данную возможность, необходимо выполнить следующую команду:

```
# setsebool -P ssh_sysadm_login off
```

12.2.4. Настройка входа по ключу

Если общение с каким-то хостом происходит очень часто, существует возможность беспарольного входа с помощью специального ключа. Данный метод безопаснее входа по паролю, так как ключ, в отличие от пароля, всегда сложнее

подобрать. Рассмотрим пример, где требуется настроить вход без пароля для пользователя ivan с хоста A на хост B под именем пользователя root.

На хосте A создайте ключ для входа:

```
$ ssh-keygen
[Enter]
[Enter]
[Enter]
```

Следующей командой скопируйте ключ на хост B:

```
$ ssh-copy-id root@B
```

Попробуйте выполнить вход на хост B. Вход должен осуществиться без ввода пароля:

```
$ ssh root@B
```

По умолчанию ssh выполняет вход от текущего пользователя. Например, если пользователь на хосте A имеет имя ivan, то команда ssh B выполнит вход на хост B также под именем ivan.

12.2.5. Сокращённые имена хостов

Для удобства системный администратор может завести короткие псевдонимы для часто используемых хостов, а также указать, какое имя пользователя по умолчанию нужно использовать. Для того чтобы добавить псевдоним для узла servnode1.example.com, необходимо добавить в файл ~/.ssh/config следующие строки:

```
Host s1
    HostName servnode1.example.com
    User root
```

После этого можно будет выполнять вход на сервер servnode1.example.com под именем пользователя root при помощи короткой команды:

```
$ ssh s1
```

Если поле User из конфигурационного файла ssh исключить, вход будет осуществляться под текущим именем пользователя, однако можно явно указать имя пользователя, используя символ «@»:

```
$ ssh root@s1
```

Можно использовать маски для группы хостов, принадлежащих одному домену. Для этого можно использовать шаблон %h в поле HostName.

```
Host node1 node2 backup
    HostName %h.example.com
```

В данном примере будут созданы псевдонимы node1 node2 и backup для узлов node1.example.com node2.example.com и backup.example.com соответственно.

12.2.6. Решение проблем

При создании ключа командой ssh-keygen новый ключ имеет имя по умолчанию id_rsa и находится в каталоге ~/.ssh. Ключ представляет собой пару файлов: личный (имя id_rsa) и публичный (имеет расширение .pub). Следует избегать публикации в открытом доступе открытого ключа и предотвращать доступ к этому ключу третьих лиц.

12.2.6.1. Вход в систему без пароля

Для удалённого входа необходимо скопировать содержимое публичного ключа на машину B в файл ~/.ssh/authorized_keys, при этом ~ означает домашний каталог пользователя, под именем которого будет осуществляться вход без пароля.

```
$ scp ~/.ssh/id_rsa.pub root@B:~/.ssh/authorized_keys
```

Если на данный хост будут осуществлять вход по ключу несколько пользователей, нужно перечислить все публичные ключи в файле authorized_keys — каждый ключ с новой строки. В данном примере файл содержит всего лишь один ключ, поэтому достаточно простого копирования публичного ключа.

Возможны проблемы с доступом по ssh при копировании файла authorized_keys из-за неправильного контекста безопасности. После того как файл был создан впервые, рекомендуется восстановить контекст безопасности:

```
$ restorecon ~/.ssh/authorized_keys
```

П р и м е ч а н и е. Рекомендуется периодически выполнять проверку, что в файле authorized_keys отсутствуют посторонние ключи.

12.2.6.2. Проблемы с DNS

По умолчанию `ssh` использует DNS для разрешения имён в локальной сети. Если в сети отсутствует DNS, он работает некорректно или DNS-сервер вышел из строя (и при этом отсутствует резервный сервер), могут возникнуть проблемы при использовании команд `ssh` и `scp`, которые выражаются в длительном (более 10 секунд) подключении.

Если при подключении по `ssh` возникают длительные задержки, временным решением проблемы может стать отключение DNS в службе `sshd`. Для этого необходимо добавить в файл `/etc/ssh/sshd_config` строчку:

```
UseDNS no
```

В любом случае источником проблемы является некорректное функционирование сервера DNS в локальной сети, которое нужно исправить.

12.2.6.3. Режим диагностики

Для диагностики проблем на стороне клиента `ssh` используйте диагностические режимы:

```
$ ssh -v
$ ssh -vv
$ ssh -vvv
```

Каждая из этих команд включает различные уровни диагностики, последний — наиболее подробный.

Для диагностики проблем на стороне сервера выполните запуск службы `sshd` от имени системного администратора с параметром `-d`. Перед этим нужно остановить службу sshd:

```
# service sshd stop
# sshd -d > log.txt 2>&1
... ожидание подключения ...
```

После успешного запуска службы (терминал был заблокирован в ожидании клиента) необходимо подключиться к данному серверу с помощью клиента `ssh`. После окончания подключения пользователя к службе `sshd`, служба завершится, оставив в файле `log.txt` всю необходимую диагностическую информацию.

12.3. Сетевая файловая служба (NFS)

12.3.1. Описание

Сетевая файловая служба/система (NFS) — протокол сетевого доступа к файловым системам. NFS предоставляет клиентам прозрачный доступ к файлам и файловой системе сервера. В отличие от FTP, протокол NFS осуществляет доступ только к тем частям файла, к которым обратился процесс, и основное достоинство его в том, что он делает этот доступ прозрачным. Это означает, что любое приложение клиента, которое может работать с локальным файлом, с таким же успехом может работать и с NFS-файлом, без каких либо модификаций самой программы.

NFS-клиенты получают доступ к файлам на NFS сервере путем отправки RPC-запросов на сервер. Это может быть реализовано с использованием обычных пользовательских процессов — а именно, NFS-клиент может быть пользовательским процессом, который осуществляет конкретные RPC-вызовы на сервер, который так же может быть пользовательским процессом.

Существует несколько версий NFS. NFS версии 2 (NFSv2) является устаревшей, но широко распространенной. NFS версии 3 (NFSv3) поддерживает асинхронную запись данных и 64-битные размеры файлов, позволяя клиентам обращаться к файлам размер которых превышает 2Гб. NFS версии 4 (NFSv4) работает через сетевые экраны, не требует службу `rpcbind`, поддерживает ACL права файлов. ОС поддерживает NFSv2, NFSv3 и NFSv4 клиентов. Все версии NFS могут использовать TCP/IP протокол для обмена данными, для NFSv4 это условие является обязательным. NFSv2 и NFSv3 могут использовать протокол UDP. При монтировании сетевой файловой системы по умолчанию используется NFSv4 если ее поддерживает удаленный сервер.

12.3.2. Настройка клиента NFS

Прежде чем обратиться к файлу на удалённой файловой системе клиент (ОС клиента) должен смонтировать её и получить от сервера указатель на неё. Монтирование NFS может производиться с помощью команды mount или с помощью одного из автоматических монтировщиков.

12.3.2.1. Монтирование файловой системы NFS командой `mount`

Монтирование осуществляется с помощью команды `mount` имеющей следующий формат:

```
# mount [-o <параметры монтирования>] <адрес сервера>:<экспортируемый каталог> <локальный каталог>
```

Адресом сервера является IP-адрес или полное доменнное имя сервера, экспортирующего файловую систему, которую требуется примонтировать. Экспортируемый каталог — корень экспортируемой файловой системы. Локальный каталог — каталог в который будет примонтирована сетевая файловая система.

Версию NFS используемую клиентом можно указать с помощью опций монтирования `nfsvers` или `vers`. По умолчанию без указания версии NFS будет использоваться NFSv4. В случае если сервер не поддерживает версию NFSv4, клиент автоматически переключится на более низкую версию NFS поддерживаемую сервером. Если указать версию NFS в параметре `nfsvers` или `vers` не поддерживаемую сервером монтирование не будет выполнено.

Основные параметры при монтировании NFS:

– `nosuid` — запрещает исполнять `setuid` программы из смонтированного каталога;

– `nodev` — запрещает использовать в качестве устройств символьные и блочные специальные файлы;

– `noexec` — запрещает запуск исполняемых файлов с примонтированной файловой системы;

– `noacl` — запрещает использование ACL на примонтированной файловой системе;

– `lock (nolock)` — разрешает блокировку NFS (по умолчанию). `nolock` отключает блокировку NFS (не запускает демон `lockd`) и удобна при работе со старыми серверами, не поддерживающими блокировку NFS. Параметры поддерживается версиями NFSv2 и NFSv3;

– `mounthost=<имя>` — имя хоста, на котором запущен демон монтирования NFS — `mountd`. Параметр поддерживается версиями NFSv2 и NFSv3;

– `mountport=n` — порт, используемый демоном `mountd`. Если параметр не указан, команда `mount` определяет порт с помощью сервиса `rpcbind`. Параметр поддерживается версиями NFSv2 и NFSv3;

– `port=n` — порт, используемый для подключения к NFS серверу. Для NFSv4 если порт не указан, то используется стандартный порт 2049. Если `n=0`, то для опре-

деления порта используется сервис `rpcbind`. Для версий NFSv2 и NFSv3 если `n=0` (по умолчанию), то NFS посылает запрос к `portmap` на сервере, чтобы определить порт;

– `rsize=n` (read block size — размер блока чтения) — количество байтов, читаемых за один раз с NFS-сервера. Стандартно — 4096;

– `wsize=n` (write block size — размер блока записи) — количество байтов, записываемых за один раз на NFS-сервер. Стандартно — 4096;

– `tcp` или `udp` — для монтирования NFS использовать протокол TCP или UDP соответственно. Параметры поддерживаются только версиями NFSv2 и NFSv3. NFSv4 всегда работает через TCP.

Атрибуты файлов, хранящиеся в индексных дескрипторах, такие как время модификации, размер, жёсткие ссылки, владелец, обычно изменяются не часто для обычных файлов и еще реже — для каталогов. Многие программы, например `ls`, обращаются к файлам только для чтения и не меняют атрибуты файлов или содержимое, но затрачивают ресурсы системы на дорогостоящие сетевые операции. Чтобы избежать ненужных затрат ресурсов, можно кэшировать данные атрибуты. Ядро использует время модификации файла, чтобы определить устарел ли кэш, сравнивая время модификации в кэше и время модификации самого файла. Кэш атрибутов периодически обновляется в соответствии с заданными параметрами:

– `ac (noac)` (attrebute cache — кэширование атрибутов) — разрешает кэширование атрибутов (по-умолчанию). Хотя параметр noac замедляет работу сервера, она позволяет избежать устаревания атрибутов, когда несколько клиентов активно записывают информацию в общую иерархию;

– `acdirmax=n` (attribute cache directory file maximum — кэширование атрибута максимум для файла каталога) — Максимальное количество секунд, которое NFS ожидает до обновления атрибутов каталога, по умолчанию 60 сек.;

– `acdirmin=n` (attribute cache directory file minimum — кэширование атрибута минимум для файла каталога) — Минимальное количество секунд, которое NFS ожидает до обновления атрибутов каталога по умолчанию 30 сек.;

– `acregmax=n` (attribute cache regular file maximum — кэширование атрибута максимум для обычного файла) — Максимальное количество секунд, которое NFS ожидает до обновления атрибутов обычного файла по умолчанию 60 сек.;

– acregmin=n (attribute cache regular file minimum — кэширование атрибута минимум для обычного файла) — Минимальное количество секунд, которое NFS ожидает до обновления атрибутов обычного файла по умолчанию 3 сек.;

– actimeo=n (attribute cache timeout — таймаут кэширования атрибутов) — заменяет значения для всех вышеуказанных параметров. Если actimeo не задан, то вышеуказанные значения принимают значения по умолчанию.

Следующие параметры управляют действиями NFS при отсутствии ответа от сервера или в случае возникновения ошибок ввода/вывода:

– fg (bg) (foreground — приоритеный режим, background — фоновый режим) — в приоритетном режиме (по умолчанию) mount возвращает код ошибки, если запрос монтирования завершается с ошибкой или по таймауту. В фоновом режиме в таком случае создается дочерний процесс, который продолжает попытки монтирования, а родительский процесс завершается с нулевым кодом возврата;

– hard (soft) — выводит на терминал сообщение «server not responding» при достижении таймаута и продолжает попытки монтирования. При заданном параметре soft — при таймауте сообщает вызвавшей операцию программе об ошибке ввода/вывода;

– retrans=n (retransmission value — значение повторной передачи) — после n малых таймаутов NFS генерирует большой таймаут (по умолчанию 3). Большой таймаут прекращает выполнение операций или выводит на консоль сообщение «server not responding», в зависимости от указания параметров hard/soft;

– retry=n (retry value — значение повторной попытки) — количество минут повторений службы NFS операций монтирования, прежде чем сдаться (по-умолчанию 10000 для фонового режима и 2 для приоритетного режима). Если n=0, то команда завершается при возникновении первой ошибки;

– timeo=n (timeout value — значение таймаута) — количество десятых долей секунды ожидания службой NFS до повторной передачи в случае RPC или малого таймаута. Это значение увеличивается при каждом таймауте до максимального значения 60 секунд или до наступления большого таймаута. В случае занятой сети, медленного сервера или при прохождении запроса через несколько маршрутизаторов или шлюзов увеличение этого значения может повысить производительность.

12.3.2.2. Монтирование файловой системы NFS с помощью файла /etc/fstab

Для монтирования файловой системы NFS при загрузке можно использовать конфигурационный файл /etc/fstab.

Общий синтаксис строки в файле /etc/fstab отвечающей за монтирование NFS следующий:

```
<адрес сервера>:<экспортируемый каталог><локальный каталог>nfs [<параметры монтирования>] 0 0
```

Параметры для монтирования NFS в конфигурационном файле /etc/fstab аналогичны одноименным параметрам при монтировании с помощью команды mount. Локальный каталог должен существовать, иначе монтирование не будет выполнено.

Для получения более подробного описания конфигурационного файла /etc/fstab выполните команду:

```
$ man fstab
```

12.3.2.3. Настройка КСЗ для работы с NFS

По умолчанию при монтировании сетевой файловой системы файлам и каталогам назначается контекст nfs_t и в безопасном режиме доступ к ним ограничивается политикой КСЗ.

В ОС определены следующие переключатели, отвечающие за работу с NFS:

– use_nfs_home_dirs — поддержка домашних каталогов NFS;

– allow_nfsd_anon_write — разрешить NFS-серверу изменять файлы, используемые для открытой передачи файлов. Каталоги и файлы должны быть отмечены как public_content_rw_t;

– nfs_export_all_rw — разрешить экспорт файлов и каталогов через NFS в режиме чтения-записи;

– nfs_export_all_ro — разрешить экспорт файлов и каталогов через NFS в режиме чтения.

Для установки выполните следующую команду:

```
# setsebool -P <переключатель> on|off
```

Параметр -P позволяет сохранить значение переключателя после перезагрузки системы. В качестве значения можно устанавливать 1, true или on для включения переключателя. Для выключения можно использовать 0, false или off.

12.3.3. Настройка сервера NFS

Существует два способа конфигурации NFS сервера: редактирование файла `/etc/exports` и использование утилиты `exportfs`.

12.3.3.1. Настройка с помощью файла `/etc/exports`

В файле `/etc/exports` описываются какие файловый системы экспортируются удаленным машинам и указываются параметры экспортирования.

Каждая строка файла `exports` имеет следующий формат:

```
<точка экспорта> <клиент>(<параметры>)
[<клиент>(<параметры>) ...]
```

Точка экспорта задает путь к эскпортируемому каталогу на сервере. Клиент задает имя одного или более клиентов или IP-адресов, разделенные пробелами, которым разрешено монтировать точку экспорта. Параметры описывают правила монтирования для клиента.

Описание клиента допускается в следующем формате:

– именем отдельного узла является полное доменное имя или IP адрес;

– несколько клиентов могут быть заданы с использованием символов «*» и «?». Например: `*.example.com`;

– подсети задаются в виде пар `<адрес IP>/<маска>`. Например: 192.168.0.0/24;

– при использовании сервера NIS сетевая группа задается в формате `@<сетевая группа>`.

Общие параметры экспортирования аналогичны файловым системам. Параметры, относящиеся к NFS:

– `auth_nlm (no_auth_nlm)` или `secure_locks (insecure_locks)` — указывает, что сервер должен требовать аутентификацию запросов на блокировку (с помощью протокола NFS Lock Manager (диспетчер блокировок NFS));

– `nohide (hide)` — если сервер экспортирует две иерархии каталогов, при этом одна вложена (примонтирована) в другую. Клиенту необходимо явно смонтировать вторую (дочернюю) иерархию, иначе точка монтирования дочерней иерархии будет выглядеть как пустой каталог. Параметр `nohide` приводит к появлению второй иерархии каталогов без явного монтирования;

– `secure (insecure)` — требует, чтобы запросы NFS поступали с защищенных портов (номер порта меньше 1024), чтобы программа без прав системного администратора не могла монтировать иерархию каталогов;

– subtree_check (no_subtree_check) — если экспортируется подкаталог файловой системы, но не вся файловая система, сервер проверяет, находится ли запрошенный файл в экспортированном подкаталоге. Отключение проверки уменьшает безопасность, но увеличивает скорость передачи данных;

– wdelay (no_wdelay) — указывает серверу задерживать выполнение запросов на запись, если ожидается последующий запрос на запись, записывая данные более большими блоками. Это повышает производительность при отправке больших очередей команд на запись. no_wdelay указывает не откладывать выполнение команды на запись, что может быть полезно, если сервер получает большое количество команд не связанных друг с другом.

Сервер NFS считает, что операционная система удаленного узла выполнила проверку подлинности пользователей и назначила им корректные идентификаторы UID и GID. Экспортирование файлов дает пользователям системы клиента такой же доступ к этим файлам, как если бы они регистрировались напрямую на сервере. Соответственно, когда клиент NFS посылает запрос серверу, сервер использует UID и GID для идентификации пользователя в локальной системе.

Следующие параметры задают правила отображения удаленных пользователей в локальных:

– root_squash (no_root_squash) — при заданном параметре root_squash, запросы от системного администратора отображаются на анонимного UID/GID, либо на пользователя, заданного в параметре anonuid/anongid;

– no_all_squash (all_squash) — не изменяет UID/GID подключающегося пользователя. Параметр all_squash задаёт отображение всех пользователей (не только системного администратора), как анонимных или заданных в параметре anonuid/anongid;

– anonuid=UID и anongid=GID — явно задает UID/GID для анонимного пользователя.

12.3.3.2. Настройка с помощью утилиты exportfs

При запуске NFS-сервера выполняется утилита exportfs, которая читает содержимое файла /etc/exports, передает управление сервису rpc.mountd (если используется NFSv2 или NFSv3) и далее сервису rpc.nfsd, который предоставляет доступ ко всем экспортируемым файловым системам удаленным пользователям. При запуске вручную утилита exportfs дает возможность администратору выборочно редактировать экспортируемые файловые системы без перезапуска сервера NFS. exportfs записывает экспортируемые файловые системы в файл

/var/lib/nfs/xtab, который обрабатывается сервисом rpc.mountd и все изменения применяются сразу после выполнения данной команды.

Параметры утилиты exportfs:

[клиент:имя-каталога] – добавить или удалить указанную файловую систему для указанного клиента);

-v — выводить подробную информацию;

-r – переэкспортировать все каталоги (синхронизировать /etc/exports и /var/lib/nfs/xtab);

-u – удалить из списка экспортируемых;

-a – добавить или удалить все файловые системы в зависимости от других параметров, переданных утилите exportfs;

-o <файловые системы> — определяет каталоги для экспортирования. Формат файловой системы аналогичен формату в файле /etc/exports. Несколько файловых систем перечисляются через запятую;

-i – не использовать /etc/exports при добавлении, только параметры текущей командной строки.

12.3.3.3. Получение информации о состоянии NFS

Получить информацию об экспортируемых файловых системах на удаленном хосте можно с помощью утилиты showmount. Утилита showmount запрашивает демон rpc.mountd на удалённом хосте о смонтированных файловых системах. По умолчанию выдаётся отсортированный список клиентов. Параметры утилиты showmount:

-a | --all – выдаётся список клиентов и точек монтирования с указанием куда клиент примонтировал каталог. Эта информация может быть не надежной;

-d | --directories – выдаётся список точек монтирования;

-e | --exports – выдаётся список экспортируемых файловых систем.

При запуске showmount без аргументов, на консоль будет выведена информация о системах, которым разрешено монтировать локальные каталоги.

Для того чтобы просмотреть статистику NFS и RPC можно использовать утилиту nfsstat.

12.3.3.4. Настройка межсетевого экрана

Для работы NFS-сервера требуется сервис rpcbind, который динамически сопоставляет порты для rpc сервисов и может вызвать проблемы в случае использования межсетевого экрана. Для того чтобы разрешить клиентам удаленное под-

ключение к NFS-ресурсам через межсетевой экран нужно отредактировать файл `/etc/sysconfig/nfs`, в котором заданы порты, на которых работают rpc службы. Требуется отредактировать следующие параметры:

– `MOUNTD_PORT=n` — номер порта, который использует `mountd` (`rpc.mountd`);

– `STATD_PORT=n` — номер порта TCP и UDP который использует `rpc.statd`;

– `LOCKD_TCPPORT=n` — номер TCP порта, который использует `nlockmgr` (`lockd`);

– `LOCKD_UDPPORT=n` — номер UDP порта, который использует `nlockmgr` (`lockd`).

Для работы с использованием межсетевого экрана на сервере NFS требуется открыть следующие порты:

1) TCP и UDP порт 2049 для NFS;

2) TCP и UDP порт 111 (`rpcbind/sunrpc`);

3) TCP и UDP порт указанный в `MOUNTD_PORT;`

4) TCP и UDP порт указанный в `STATD_PORT;`

5) TCP порт указанный в `LOCKD_TCPPORT;`

6) UDP порт указанный в `LOCKD_UDPPORT.`

Для работы только с версией NFSv4 достаточно открыть TCP порт 2049, а также TCP и UDP порт 111.

12.3.3.5. Запуск сервера NFS

Запустить NFS-сервер можно с помощью следующей команды:

```
# service nfs start
```

Для остановки сервера выполните следующую команду:

```
# service nfs stop
```

Для того чтобы NFS-сервер запускался при загрузке системы, требуется добавить запуск сервера в автозагрузку. Для этого выполните следующую команду:

```
# chkconfig nfs on
```

Для того чтобы убрать запуск NFS-сервера при загрузке, требуется выполнить следующую команду:

```
# chkconfig nfs off
```

12.4. Сервер инфраструктуры LDAP

12.4.1. Описание

LDAP (англ. Lightweight Directory Access Protocol — «облегчённый протокол доступа к каталогам») — протокол прикладного уровня для доступа к службе каталогов X.500. LDAP — относительно простой протокол, использующий TCP/IP и позволяющий производить операции аутентификации (bind), поиска (search) и сравнения (compare), а также операции добавления, изменения или удаления записей. Обычно LDAP-сервер принимает входящие соединения на порт 389 по протоколам TCP или UDP. Для LDAP-сеансов, инкапсулированных в SSL, обычно используется порт 636.

Всякая запись в каталоге LDAP состоит из одного или нескольких атрибутов и обладает уникальным именем (DN — англ. Distinguished Name). Уникальное имя может выглядеть, например, следующим образом: «`cn=Иван Петров`, `ou=Сотрудники`, `dc=example`, `dc=com`». Уникальное имя состоит из одного или нескольких относительных уникальных имен (RDN — англ. Relative Distinguished Name), разделённых запятой. Относительное уникальное имя имеет вид `ИмяАтрибута=значение`. На одном уровне каталога не может существовать двух записей с одинаковыми относительными уникальными именами. В силу такой структуры уникального имени записи в каталоге LDAP можно легко представить в виде дерева.

Запись может состоять только из тех атрибутов, которые определены в описании класса записи (object class), которые, в свою очередь, объединены в схемы (schema). В схеме определено, какие атрибуты являются для данного класса обязательными, а какие — необязательными. Также схема определяет тип и правила сравнения атрибутов. Каждый атрибут записи может хранить несколько значений.

12.4.2. Настройка клиента LDAP

В качестве примера рассмотрим настройку авторизации пользователей в системе с использованием сервера LDAP. Для авторизации пользователей в системе с использованием сервера LDAP в ОС должен быть установлен пакет `pam_ldap`. Для его установки выполните следующую команду:

```
# yum install pam_ldap
```

По умолчанию в ОС используется сервис `sssd` (System Security Services Daemon — системный сервис безопасности), который запрещает установку незащищенных соединений с сервером LDAP.

Если сеть является доверенной и требуется установить незащищенное соединение с сервером LDAP, то вместо сервиса `sssd` должен использоваться сервис `nslcd` из пакета `nss-pam-ldapd`.

Для его установки выполните следующую команду:

```
# yum install nss-pam-ldapd
```

Для разрешения аутентификации с использованием незащищенного соединения с сервером LDAP выполните следующую команду:

```
# authconfig --enableldap --enableldapauth
--ldapserver=<адрес сервера> --ldapbasedn="dc=example,dc=com"
--enablemkhomedir --disablesssd --disablesssdauth
--enableforcelegacy --disableldaptls --update
```

При выполнении данной команды сервис `sssd` будет отключен и вместо него будет использоваться сервис `nslcd`. Для получения более подробной информации об утилите настройки аутентификации выполните:

```
$ man authconfig
```

или

```
$ authconfig --help
```

Далее потребуется перезагрузить систему и после загрузки ОС можно авторизоваться пользователем, зарегистрированным на сервере LDAP.

12.4.3. Настройка защищенного соединения на клиенте LDAP

Существует два варианта настройки защищенного соединения с сервером LDAP: с использованием TLS и с использованием SASL.

При использовании TLS используется 389 порт, схемой URI является `ldap`. При использовании SASL используется 636 порт, схемой URI является `ldaps`. Для установки защищенного соединения с сервером LDAP он должен иметь сертификат. Сертификат может быть либо самоподписанный, либо выданный центром сертификации.

12.4.3.1. Настройка LDAP клиента для использования TLS

В качестве примера рассмотрим настройку клиента LDAP для использования TLS и самоподписанного сертификата LDAP сервера.

По умолчанию в ОС используется сервис `sssd` (System Security Services Daemon — системный сервис безопасности), который запрещает использование самоподписанного сертификата сервера LDAP. Если требуется установить соединение с сервером LDAP с использованием самоподписанного сертификата, то вместо сервиса `sssd` должен использоваться сервис `nslcd` из пакета `nss-pam-ldapd`.

Для его установки выполните следующую команду:

```
# yum install nss-pam-ldapd
```

Для разрешения аутентификации с использованием TLS соединения с сервером LDAP выполните следующую команду:

```
# authconfig --enableldap --enableldapauth
--ldapserver=<адрес сервера> --ldapbasedn="dc=example,dc=com"
--enablemkhomedir --disablesssd --disablesssdauth
--enableforcelegacy --enableldaptls --enableldapstarttls
--update
```

При выполнении данной команды сервис `sssd` будет отключён и вместо него будет использоваться сервис `nslcd`. Для изменения настроек сервиса `nslcd` требуется отредактировать файл `/etc/nslcd.conf`

Для этого выполните следующие действия:

– запретите поиск сертификатов для авторизации сервера:

```
# sed -i.bak 's/\(.*tls_cacertdir.*\)/#\1/g' /etc/nslcd.conf
```

– разрешите устанавливать соединение без проверки сертификата:

```
# echo "tls_reqcert allow" >> /etc/nscld.conf
```

Для получения более подробной информации о настройке сервиса nslcd выполните:

```
$ man nslcd.conf
```

Далее потребуется перезагрузить систему и после загрузки ОС можно авторизоваться пользователем, зарегистрированным на сервере LDAP.

12.4.3.2. Настройка LDAP клиента для использования SASL

В качестве примера рассмотрим настройку клиента LDAP для использования SASL и сертификата, подписанного центром сертификации.

Для разрешения аутентификации с использованием SASL соединения с сервером LDAP выполните следующие действия:

– скопируйте сертификат с сервера LDAP, например, с помощью SSH:

```
# scp <адрес сервера>/etc/pki/CA/cacert.pem
/etc/openldap/cacert.pem
```

– настройте сервис `sssd` для авторизации выполнив команду:

```
# authconfig --enableldap --enableldapauth
--ldapserver=<адрес сервера> --ldapbasedn="dc=example,dc=com"
--enablemkhomedir --enablesssd --enablesssdauth
--disableforcelegacy --enableldaptls --disableldapstarttls
--update
```

Параметр адрес сервера в данной команде при использовании SASL должен начинаться с «`ldaps://`».

Для получения более подробной информации о настройке сервиса `sssd` выполните:

```
$ man sssd.conf
```

Далее потребуется перезагрузить систему и после загрузки ОС можно авторизоваться пользователем, зарегистрированным на сервере LDAP.

12.4.4. Настройка сервера LDAP

12.4.4.1. Начальная настройка сервера LDAP

Для работы сервера в ОС должен быть установлен пакет `openldap-servers`.

Для его установки выполните следующую команду:

```
# yum install openldap-servers
```

Для начальной настройки сервера выполните следующие команды (выполнение данных команд приведет к удалению существующей базы данных LDAP сервера):

```
# service slapd stop
# rm -rf /etc/openldap/slapd.d/*
# rm -rf /var/lib/ldap/*
# cp /usr/share/doc/openldap-servers/DB_CONFIG.example
/var/lib/ldap/DB_CONFIG
# cat /usr/share/openldap-servers/slapd.conf.obsolete | sed
's/dc=my-domain/dc=example/g' > /etc/openldap/slapd.example.conf
```

Задайте пароль администратора LDAP, выполнив следующие команды:

```
# PASSWORD=‘slappasswd -s <пароль> -h {md5}‘; sh -c ”sed -i.bak ’s/^#[ \t]*rootpw[ \t]*secret.*/rootpw $PASSWORD/’ /etc/openldap/slapd.example.conf”
# echo ”” | slapadd -f /etc/openldap/slapd.example.conf
# slaptest -f /etc/openldap/slapd.example.conf -F /etc/openldap/slapd.d
# rm -f /etc/openldap/slapd.example.conf /etc/openldap/slapd.example.conf.bak
```

Для работы с использованием межсетевого экрана на сервере LDAP требуется открыть 389 TCP и UDP порты.

12.4.4.2. Подготовка базы данных сервера LDAP

В качестве примера рассмотрим создание административной структуры пользователей и групп в базе данных сервера LDAP.

Для добавления информации в базу данных LDAP сервера в ОС должен быть установлен пакет `openldap-clients`.

Для его установки выполните следующую команду:

```
# yum install openldap-clients
```

Для соединения с LDAP сервером отредактируйте файл `/etc/openldap/ldap.conf` следующим образом:

– установите параметр `BASE` в `dc=example,dc=com`;

– в параметре URI укажите адрес LDAP сервера.

Для получения более подробной информации по настройке конфигурационного файла выполните команду:

```
$ man ldap.conf
```

Для начального наполнения базы данных сервера LDAP выполните следующие действия:

1) создайте файл `example.com.ldif` следующего содержания:

```
dn: dc=example,dc=com
objectclass: dcObject
objectclass: organization
o: Example
dc: example
```

2) создайте файл `manager.example.com.ldif` следующего содержания:

```
dn: cn=Manager,dc=example,dc=com
objectclass: organizationalRole
cn: Manager
```

3) создайте файл `users.example.com.ldif` следующего содержания:

```
dn: ou=users,dc=example,dc=com
changetype: add
objectclass: top
objectclass: organizationalUnit
ou: users
```

4) создайте файл `groups.example.com.ldif` следующего содержания:

```
dn: ou=groups,dc=example,dc=com
changetype: add
objectclass: top
objectclass: organizationalUnit
ou: groups
```

5) инициализируйте базу данных сервера LDAP, выполнив следующие команды:

```
# slapadd -l example.com.ldif
# slapadd -l manager.example.com.ldif
```

6) установите права доступа к базе данных LDAP выполнив следующие команды:

```
# chown -R ldap:ldap /var/lib/ldap
# chown -R ldap:ldap /etc/openldap/slapd.d
```

7) запустите сервер LDAP, выполнив следующую команду:

```
# run_init /etc/init.d/slapd start
```

8) добавьте записи структур пользователей и групп в базу данных сервера LDAP выполнив следующие команды:

```
# ldapadd -x -D 'cn=admin,dc=example,dc=com' -W -f
users.example.com.ldif
# ldapadd -x -D 'cn=admin,dc=example,dc=com' -W -f
groups.example.com.ldif
```

Для получения более подробной информации по утилите `ldapadd` выполните:

```
$ man ldapadd
```

12.4.4.3. Добавление данных на сервер LDAP

В качестве примера рассмотрим добавление существующих пользователей в базу данных сервера LDAP.

Для добавления существующих пользователей в базу данных сервера LDAP в ОС должен быть установлен пакет `migrationtools`. Для его установки выполните следующую команду: `#yum install migrationtools`. Отредактируйте файл `/usr/share/migrationtools/migrate_common.ph` выполнив следующие команды:

```
    # sed -i.bak 's/\($DEFAULT_MAIL_DOMAIN[ \t]*=[
\t]*\"\)\(.*\)\(\";\)/\1example.com\3/g'
/usr/share/migrationtools/migrate_common.ph
    # sed -i 's/\($DEFAULT_BASE[ \t]*=[
\t]*\"\)\(.*\)\(\";\)/\1dc=example,dc=com\3/g'
/usr/share/migrationtools/migrate_common.ph
    # sed -i 's/\([ \t]*$NAMING_CONTEXT{'\''passwd'\''}[ \t]*=[
\t]*\"\)\(.*\)\(\";\)/\1ou=users\3/g
/usr/share/migrationtools/migrate_common.ph
    # sed -i 's/\([ \t]*$NAMING_CONTEXT{'\''groups'\''}[ \t]*=[
\t]*\"\)\(.*\)\(\";\)/\1ou=groups\3/g
/usr/share/migrationtools/migrate_common.ph
```

Создайте пользователя `user1` и задайте ему пароль, выполнив следующие команды:

```
    # useradd user1
    # passwd user1
```

Для добавления существующего пользователя `user1` в базу данных сервера LDAP требуется выполнить следующие действия:

1) создайте временные файлы, содержащие информацию о пользователе user1, выполнив следующие команды:

```
    # grep user1 /etc/passwd > user1.passwd.line
    # grep user1 /etc/group > user1.group.line
```

2) экспортируйте файлы, содержащие информацию о пользователе `user1`, в формат LDIF выполнив следующие команды:

```
    # /usr/share/migrationtools/migrate_passwd.pl
user1.passwd.line user1.passwd.ldif
```

```
# /usr/share/migrationtools/migrate_group.pl
user1.group.line user1.group.ldif
```

3) добавьте информацию о пользователе `user1` в базу данных сервера LDAP, выполнив следующие команды:

```
# ldapadd -x -D ”cn=Manager,dc=example,dc=com” -W -f
user1.passwd.ldif
# ldapadd -x -D ”cn=Manager,dc=example,dc=com” -W -f
user1.group.ldif
```

4) удалите пользователя `user1` из системы, выполнив следующие команды:

```
# userdel user1
# groupdel user1
```

5) удалите временные файлы, созданные при добавлении пользователя `user1`, выполнив следующую команду:

```
# rm -f user1.passwd.line user1.group.line user1.passwd.ldif
user1.group.ldif
```

12.4.5. Настройка защищенного соединения на сервере LDAP

Существует два варианта настройки защищенного соединения на сервере LDAP: с использованием TLS, с использованием SASL.

При использовании TLS используется 389 порт, схемой URI является `ldap://`. При использовании SASL используется 636 порт, схемой URI является `ldaps://`. Для установки защищенного соединения сервер LDAP должен иметь сертификат. Сертификат может быть либо самоподписанный, либо выданный центром сертификации.

12.4.5.1. Настройка LDAP сервера для использования TLS

В качестве примера рассмотрим настройку сервера LDAP для использования TLS и самоподписанного сертификата. Для настройки сервера LDAP выполните следующие действия:

1) остановите работу сервера LDAP:

```
# service slapd stop
```

2) для динамического изменения конфигурации LDAP сервера разрешите IPC соединения с сервером LDAP, отредактировав файл: `/etc/sysconfig/ldap`.

```
# sed -i.bak ’s/.*SLAPD_LDAPI[ \t]*=[
\t]*no/SLAPD_LDAPI=yes/’ /etc/sysconfig/ldap
```

3) задайте пароль на изменение общих настроек LDAP сервера:

```
# PASSWORD=‘slappasswd -s <пароль> -h {md5}‘; echo -e
”olcRootDN: cn=config\nolcRootPW: $PASSWORD” >>
/etc/openldap/slapd.d/cn=config/olcDatabase={0}config.ldif
```

4) для добавления параметров сертификатов создайте файл `add.cert.conf.ldif` следующего содержания:

```
dn: cn=config
changetype: modify
add: olcTLSCertificateFile
olcTLSCertificateFile: /etc/pki/tls/certs/slapdcert.pem
-
add: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /etc/pki/tls/certs/slapdkey.pem
-
```

5) запустите сервер LDAP:

```
# run_init /etc/init.d/slapd start
```

6) добавьте параметры сертификатов на сервер LDAP:

```
# ldapmodify -D ”cn=config” -H ldapi://<адрес сервера> -f
add.cert.conf.ldif -W
```

7) остановите работу сервера LDAP:

```
# service slapd stop
```

8) создайте самоподписанный сертификат:

```
# openssl req -new -x509 -nodes -out
/etc/pki/tls/certs/slapdcert.pem -keyout
/etc/pki/tls/certs/slapdkey.pem -days 365
```

При создании сертификата необходимо заполнить несколько параметров:
CountryName (2 letter code) [XX]: - 2 символа кода страны (RU);
State or Province Name (full name) []: - округ или область;
LocalityName (eg, city) [Default City]: - город;
Organization Name (eg, company) [Default Company Ltd]: - название организации;
Organizational Unit Name (eg, section) []: - название подразделения;
Common Name (eg, your name or your server’s hostname) []: - имя сервера;
Email Address []: - адрес электронной почты;

9) установите права доступа к сертификатам сервера LDAP:

```
# chown -Rf root:ldap /etc/pki/tls/certs/slapdcert.pem
/etc/pki/tls/certs/slapdkey.pem
# chmod -Rf 750 /etc/pki/tls/certs/slapdkey.pem
```

10) запустите сервер LDAP:

```
# run_init /etc/init.d/slapd start
```

12.4.5.2. Настройка LDAP сервера для использования SASL

В качестве примера рассмотрим настройку сервера LDAP для использования SASL и сертификата, подписанного центром сертификации.

Для настройки сервера LDAP выполните следующие действия:

1) остановите работу сервера LDAP:

```
# service slapd stop
```

2) для использования SASL отредактируйте файл `/etc/sysconfig/ldap` выполнив следующие команды:

```
# sed -i.bak 's/.*SLAPD_LDAP[ \t]*=[
\t]*yes/SLAPD_LDAPI=no/' /etc/sysconfig/ldap
# sed -i 's/.*SLAPD_LDAPS[ \t]*=[ \t]*no/SLAPD_LDAPI=yes/'
/etc/sysconfig/ldap
```

3) создайте сертификат, подписанный центром сертификации, выполнив следующие действия:

```
# rm
/etc/pki/CA/{cacert.pem,serial,crlnumber,cakey.pem,index.txt}
# rm /etc/pki/tls/{server.example.com.csr}
# cat /dev/null > /etc/pki/CA/index.txt
# echo "01" > /etc/pki/CA/serial
# echo "01" > /etc/pki/CA/crlnumber
# openssl req -new -x509 -extensions v3_ca -keyout
/etc/pki/CA/private/cakey.pem -out /etc/pki/CA/cacert.pem
```

При выполнении данной команды необходимо заполнить несколько параметров:

– Enter PEM pass phrase: - ключевое слово, которое будет использовано в дальнейшем (требуется запомнить его)

– Verifying - Enter PEM pass phrase: - если длина ключевого слова больше 3 символов, то его ввод будет запрошен повторно

– CountryName (2 letter code) [XX]: - 2 символа кода страны (RU)

– State or Province Name (full name) []: - округ или область

– LocalityName (eg, city) [Default City]: - город

– Organization Name (eg, company) [Default Company Ltd]: - название организации

– Organizational Unit Name (eg, section) []: - название подразделения

– Common Name (eg, your name or your server's hostname) []: - имя сервера

– Email Address []: - адрес электронной почты

Параметр Common Name должен быть обязательно заполнен, остальные параметры не являются обязательными.

```
# openssl req -new -nodes -keyout
/etc/pki/tls/private/server.example.com.key -out
/etc/pki/tls/server.example.com.csr
```

При выполнении данной команды необходимо заполнить несколько параметров (аналогично параметрам запрошенным при выполнении предыдущей команды):

– CountryName (2 letter code) [XX]: - 2 символа кода страны (RU)

– State or Province Name (full name) []: - округ или область

– LocalityName (eg, city) [Default City]: - город

– Organization Name (eg, company) [Default Company Ltd]: - название организации

– Organizational Unit Name (eg, section) []: - название подразделения

– Common Name (eg, your name or your server's hostname) []: - имя сервера

– Email Address []: - адрес электронной почты

– A challenge password []: - проверочный пароль

– An optional company name []: - название организации

Параметр Common Name должен быть обязательно заполнен, остальные параметры не являются обязательными.

```
# openssl ca -policy policy_anything -out
/etc/pki/CA/certs/server.example.com.crt -infiles
/etc/pki/tls/server.example.com.csr
```

При выполнении данной команды потребуется ввести ключевое слово из пункта выше.

```
# cp -f /etc/pki/CA/cacert.pem /etc/pki/tls/certs
# openssl x509 -noout -hash -in
/etc/pki/tls/certs/cacert.pem
```

```
    # HASH=$( openssl x509 -noout -hash -in
/etc/pki/tls/certs/cacert.pem )#ln -s
/etc/pki/tls/certs/cacert.pem /etc/pki/tls/certs/${HASH}.0
    # cp -f /etc/pki/tls/private/server.example.com.key
/etc/pki/tls/certs/slapdkey.pem
    # cp -f /etc/pki/tls/CA/certs/server.example.com.csr
/etc/pki/tls/certs/slapdcert.pem
```

4) запустите сервер LDAP:

```
    # run_init /etc/init.d/slapd start
```

12.5. Тонкий клиент

В данном разделе описывается настройка тонкого клиента.

12.5.1. Настройка клиента

Вставьте диск с ОС и установите пакет `openssh-askpass` выполнив команды:

```
    # mkdir -p /misc/cd
    # mount /dev/sr0 /misc/cd
    # yum install openssh-askpass
```

Отредактируйте файл /etc/lxdm/lxdm.conf изменив строку session=/usr/bin/xfce4-session на строку:

```
    session = ssh -X <IP> xfce4-session
```

где:

`<IP>` — IP адрес сервера

Отредактируйте файл /etc/xdg/xfce4/xinitrc изменив строку xfce4-session на строку:

```
    env SSH_ASKPASS="" setsid ssh -X <USER>@<IP> xfce4-session
```

где:

`<USER>` — имя пользователя подключающегося к серверу;

`<IP>` — IP-адрес сервера.

Перезагрузите систему.

Введите локальный пароль пользователя, затем введите пароль пользователя на удаленном сервере. После загрузки на экране появится рабочий стол удаленного пользователя.

12.5.2. Настройка сервера

Для подключения к серверу в принудительном режиме создайте файл z.te следующего содержания:

```
module zsshx 1.0;
require{
type xserver_port_t;
attribute wm_domain;
attribute x_userdomain;
attribute x_domain;
class tcp_soclet { name_connect };
class peer { recv };
}
allow wm_domain xserver_port_t:tcp_socket name_connect;
allow wm_domain x_userdomain:peer recv;
allow x_userdomain wm_domain:peer recv;
allow x_userdomain x_domain:peer recv;
allow x_domain x_userdomain:peer recv;
```

Для загрузки модуля политики выполните следующие команды:

```
# checkmodule -m -M -o z.mod z.te
# semodule_package -m z.mod -o z.pp
# semodule -i z.pp
```

13. РЕШЕНИЕ ПРОБЛЕМ

В данном разделе перечислены основные проблемы, которые могут возникнуть в процессе настройки и/или эксплуатации операционной системы.

13.1. Диагностика потребления ресурсов

В случае, когда ОС начинает реагировать на команды пользователя непривычно долго, выполните от имени системного администратора следующий сценарий диагностики:

1) выполните команду `free` для просмотра статистики использования памяти и файла подкачки:

```
# free -m
```

В случае, когда поле `used` значительно превышает поле `free`, низкая производительность может оказаться следствием чрезмерного потребления памяти каким-либо из процессов;

2) выполните команду `df` для просмотра статистики использования дисков. Если какой-либо из дисков заполнен более чем на 95% — это может приводить к значительному снижению производительности операций ввода-вывода. Наиболее частая причина внезапного исчезновения свободного места на жёстком диске — переполнение каталогов `/var/log` и `/tmp`;

3) выполните команду `top` для просмотра списка процессов, которые наиболее интенсивно используют центральный процессор. Данная строка показывает, что центральный процессор нагружен только наполовину:

```
Cpu(s): 52.6%us
```

Стоит учитывать, что на многоядерных процессорах степень загруженности зависит от числа процессоров. Например, на четырёхъядерной машине в списке процессов потребление процессорного времени будет обозначаться как 200%, что означает полную загрузку только двух ядер из четырёх, при этом поле `Cpu(s)` будет отображать загрузку примерно 50%. Для того чтобы найти процесс, который потребляет больше всего оперативной памяти, нажмите сочетание клавиш [Shift+M]. Имеет смысл ориентироваться на объём занимаемой резидентной памяти (столбец `RES`);

4) для того чтобы уничтожить проблемный процесс, выполните команду:

```
# kill <PID>
```

или

```
# killall <имя>
```

Выполняя команду `killall` стоит иметь в виду, что будут уничтожены все процессы с таким же именем. Если после выполнения команды `killall` процесс продолжает работу («завис»), можно принудительно уничтожить процесс:

```
# kill -9 <PID>
```

или

```
# killall -9 <PID>
```

ВНИМАНИЕ! ВСЕ НЕСОХРАНЁННЫЕ ДАННЫЕ ПРОЦЕССА БУДУТ ПОТЕРЯНЫ. ИСПОЛЬЗУЙТЕ ДАННЫЙ МЕТОД ТОЛЬКО В КРАЙНЕМ СЛУЧАЕ.

13.2. Диагностика ошибок

Если при работе ОС возникает ошибка, характер которой не удаётся определить, наиболее результативным способом диагностики является чтение журналов. Большинство файлов журналов системы можно найти в каталоге `/var/log`. Наиболее важные журналы из данного каталога, которые следует просмотреть в первую очередь, перечислены ниже.

`messages` журнал, содержащий практически все сообщения об ошибках.

`secure` журнал удалённой аутентификации. Содержит все сообщения о попытках проникнуть на компьютер извне, например, с использованием `ssh`.

`audit/audit.log` журнал аудита. Содержит сообщения, имеющий отношение к КСЗ.

`dmesg` журнал сообщений ядра. Используется при возникновении проблем с ядром ОС или драйверами и устройствами.

`boot.log` журнал загрузки системы — содержит подробную информацию о процедуре старта ОС.

`cron` журнал планировщика, содержит результаты запуска периодических сценариев.

`yum.log` журнал установки/удаления/обновления пакетов. Позволяет отслеживать, какие были внесены изменения в состав пакетов дистрибутива.

Подробную справочную информацию по мониторингу и журналированию см. в разделе 9.1.

13.3. Восстановление системы из режима обслуживания

В случае нарушения работы, программного или аппаратного сбоя или ошибке администратора при настройке системы, после перезагрузки операционная система переходит в режим обслуживания (maintenance mode). В этот режим система переходит автоматически при обнаружении проблем, на экран выводится краткое описание ошибки и предложение о вводе пароля системного администратора. Для перезагрузки системы предлагается нажать сочетание клавиш [Ctrl+D].

Для того, чтобы приступить к разрешению проблемы, введите пароль системного администратора. После ввода пароля появится приглашение командной строки и можно приступать к исправлению проблемы.

Наиболее частые причины аварийного перехода системы в режим обслуживания:

– нарушена целостность файловой системы;

– неправильная конфигурация жестких дисков (например, ошибка в файле `/etc/fstab`).

В режиме обслуживания система загружается с примонтированной корневой файловой системой в режиме «только чтение». Для того, чтобы получить возможность изменения конфигурационных файлов на жёстком диске, необходимо выполнить перед этим следующую команду:

```
# mount -o remount,rw /
```

После этого корневая файловая система будет примонтирована в режиме «чтение/запись» и станет возможной правка конфигурационных файлов.

Если проблема заключается в ошибке на жестком диске, необходимо выполнить команду `fsck`, указав проблемный раздел. Раздел, на котором произошла ошибка, обычно всегда выводится в терминал при входе в режим обслуживания. Выполните команду:

```
# fsck <путь до раздела>
```

Здесь, `<путь до раздела>` — полный путь до устройства жёсткого диска, на котором произошла ошибка.

Пример для раздела диска, размеченного с помощью логических томов:

```
# fsck /dev/mapper/VolGroup-lv_var_log
```

Пример для диска, размеченного классическим образом (по разделам):

```
# fsck /dev/sda1
```

13.4. Проверка и восстановление поврежденных файлов

В процессе неправильной настройки или эксплуатации операционной системы может возникнуть необходимость диагностики и/или восстановления как бинарных модулей системы, так и конфигурационных файлов.

Диагностика повреждённых файлов осуществляется командой `rpm`, а восстановление — командой `yum reinstall`. Обе команды могут быть запущены только от имени системного администратора. Данный инструмент является одним из наиболее эффективных средств диагностики, так как позволяет обнаружить:

– изменения в системных конфигурационных файлах, которые привели к неработоспособности системы/службы/приложения;

– удалённые или повреждённые системные файлы: библиотеки, приложения, ресурсы;

– изменения дискреционных прав доступа по умолчанию на файлы и каталоги, которые могут приводить к неработоспособности некоторых приложений и служб.

13.4.1. Локализация повреждённых файлов

Локализовать поврежденные файлы и характер повреждения позволяет встроенный механизм верификации (проверки). Для запуска механизма проверки необходимо запустить команду `rpm`.

Общая форма команды проверки `rpm` приведена ниже:

```
rpm -V|--verify [<параметры выбора>] [<параметры проверки>]
```

Операция проверки пакета сравнивает информацию о файлах установленных из пакета с информацией о них из метаданных пакета, хранимых в базе данных `rpm`. Среди прочего при проверке сравниваются размер, контрольная сумма, права доступа, тип, владелец и группа каждого файла. Любые расхождения будут отображены.

Файлы, которые не были установлены вместе с пакетом, например, файлы документации, исключенные при помощи параметра `--excludedocs`, будут проигнорированы без предупреждения.

Параметры выбора пакетов являются аналогичными запросу пакетов. Описание команды `rpm` приведено в разделе 5.3.

Параметры, уникальные для режима проверки, приведены ниже.

`--nodeps` не выполнять проверку зависимостей пакетов.

`--nodigest` не проверять при чтении дайджест пакета или заголовка.

`--nofiles` не проверять атрибуты файлов пакетов.

`--noscripts` не выполнять скриптлет `%verifyscript` (если существует).

`--nosignature` не проверять при чтении подпись пакета или заголовка при чтении.

`--nolinkto` не проверять изменение символьных ссылок (атрибут L).

`--nomd5` не проверять изменение контрольной суммы (атрибут 5).

`--nosize` не проверять изменение размера файла (атрибут S).

`--nouser` не проверять изменение владельца файла (атрибут U).

`--nogroup` не проверять изменение группы владельца файла (атрибут G).

`--nomtime` не проверять изменение времени изменения файла (атрибут T).

`--nomode` не проверять изменение прав доступа файла (атрибут M).

`--nordev` не проверять изменение номера устройства (атрибут D).

Формат вывода представляет собой строку из 8 символов и маркера из заголовка пакета, за которыми следует имя файла. Возможные маркеры атрибутов:

– `c %config` конфигурационный файл;

– `d %doc` файл документации;

– `g %ghost` «файл-призрак» (т. е. содержимое файла не включено в состав пакета);

– `l %license` файл с лицензией;

– `r %readme` файл readme.

Каждый из восьми символов отражает результат проверки атрибута(ов) файлов с значением того же атрибута, записанного в базе данных. Символ «.» (точка) означает, что проверка прошла, а символ «?» (вопросительный знак) означает, что проверка не может быть выполнена (например, права доступа к файлу не позволяют провести чтение). В противном случае будут отображены символы (для привлечения внимания выделены жирным), показывающие сбой проверки соответствующего `--verify` теста:

`S` — размер файла отличается;

`M` — режим доступа отличается (включая права доступа и тип файла);

`5` — отличается контрольная (MD5) сумма (как правило, это означает отличие в содержимом файла);

`D` — отличается старший/младший номер файла устройства;

`L` — отличается путь ссылки;

`U` — отличается владелец;

G — отличается группа владельца;

T — отличается время изменения.

Если проблемный пакет известен, лучше производить проверку только для одного конкретного пакета, так как объем выводимой информации сильно уменьшится. Например, проверка пакета kernel (ядро операционной системы):

```
# rpm -V kernel
```

Если команда ничего не выводит на экран, это означает, что все файлы в данном пакете не изменились с момента установки операционной системы.

В случае обнаружения файлов, отличающихся от предустановленных, rpm выведет их на экран:

```
# rpm -V httpd
S.5....T.  c /etc/httpd/conf/httpd.conf
```

В данном примере видно, что конфигурационный файл /etc/httpd/conf/httpd.conf (о чём свидетельствует символ c) имеет другой размер (S), отличается по содержимому (5) и времени изменения (T). Это стандартный набор изменений для конфигурационного файла: системные администраторы их часто меняют, чтобы изменить поведение веб-сервера и настроить программу под свои нужды, при этом меняется содержимое, размер и временная метка файла.

Можно также выполнять фильтрацию командой grep, если изменений очень много:

```
# rpm -V apache | grep ^??5
```

В данном примере будут выведены только те файлы, у которых изменилось содержимое. Контрольная сумма наиболее точно определяет изменения в файле, так как изменения в размере при изменения файла может и не происходить.

13.4.2. Восстановление файлов

Иногда системный администратор в процессе настройки системы может изменить конфигурационный файл таким образом, что необходимая или связанная с данным файлом служба перестала работать. При этом нигде не сохранилась резервная копия данного файла.

В случае возникновения подобных проблем системный администратор может прибегнуть к процедуре восстановления файлов с диска. Для этого потребуется вставить DVD с ОС в устройство чтения дисков.

Рассмотрим ситуацию: администратор редактировал файл `/etc/httpd/conf/httpd.conf` для настройки веб-сервера и в определённый момент изменил его таким образом, что веб-сервер больше не работает.

В первую очередь, нужно установить, к какому пакету принадлежит файл:

```
# rpm -qf /etc/httpd/conf/httpd.conf
httpd-2.2.15-9.el6.x86_64
```

Вывод команды сообщает, что файл относится к пакету `httpd` (версию и прочую информацию можно опустить).

Следующий этап — проверка данного пакета:

```
# rpm -V httpd
S.5....T.  c /etc/httpd/conf/httpd.conf
```

Вывод данной команды подтверждает наличие изменений в конфигурационном файле и только в нём. Перед восстановлением нужно удалить исходный файл:

```
# rm /etc/httpd/conf/httpd.conf
```

Так как удаление файла приведёт к невозможности его восстановить, рекомендуется вместо удаления всегда переименовывать файл:

```
# mv /etc/httpd/conf/httpd.conf{,.bak}
```

Таким образом, будет сохранена резервная копия, а впоследствии можно будет сравнить два файла конфигурации, чтобы точнее диагностировать проблему.

Восстановить удалённый (переименованный) файл можно следующей командой:

```
# yum reinstall -y httpd
```

Дождитесь окончания работы команды `yum`, после чего убедитесь, что файл восстановлен:

```
# rpm -V httpd
```

Вывод команды должен быть пуст.

После того как файл был восстановлен, следует приступить к анализу изменений конфигурационного файла, которые привели к ошибке:

```
# diff /etc/httpd/conf/httpd.conf{,.bak}
```

По завершении анализа можно будет перенести настройки из резервной копии в основной конфигурационный файл с учётом исправления ошибок.

Аналогичным образом можно восстановить любой повреждённый файл в системе, включая программы, библиотеки и т. п.

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ

КСЗ — комплекс средств защиты
НСД — несанкционированный доступ
ОС — операционная система
ПО — программное обеспечение
ПРД — правила разграничения доступа
ФС — файловая система