В число основных задач современных вычислительных систем входит обработка текстовой информации, как в виде простого текста, так и в виде текста с форматированием. Хотя форматированный текст на персональных компьютерах обычно представляется в формате двоичных файлов, в последнее время намечается тенденция отказа от таких (часто закрытых) двоичных форматов и перехода к использованию основанных на обычном тексте языков разметки документов. Операционные системы *nix изначально разрабатывались для обработки текстовой информации, и обладают большим набором мощных и универсальных инструментов работы с текстами. Одним из таких инструментов являются регулярные выражения, примеры применения которых также рассматриваются в данной работе.

Структура файловой системы.

Для хранения данных в настоящий момент используются различные устройства — накопители на жестких и гибких магнитных дисках, накопители на микросхемах Flash-памяти, накопители на оптических носителях форматов CD, DVD, Blu-ray, и т.п. С точки зрения операционных систем, всё это — устройства с блочным вводом-выводом, которые далее мы будем обобщённо называть дисками.

Как правило, доступное для хранения информации место на дисках разбивается на разделы. В рамках каждого из разделов создаётся файловая система, позволяющая управлять размещением на дисках отдельных файлов. Это требование не является жёстким, возможно создание файловых систем непосредственно на дисках, без разбиения их на отдельные разделы. Кроме того, возможно хранение информации на дисках и без создания файловых систем — например, крупные системы управления базами данных (СУБД) могут сами управлять размещением баз данных на дисках, без использования промежуточных звеньев в виде файлов и файловых систем.

Задачей файловой системы является обеспечение эффективного выделения пространства для хранения данных, ведение списка файлов и каталогов, эффективный поиск файлов в каталогах и т.д. Существует большое количество файловых систем, обладающих теми или иными характеристиками. Выбор файловой системы для носителя данных зависит от конкретного случая. Операционные системы могут одновременно управлять несколькими устройствами хранения данных с разными файловыми системами на них.

К основным поддерживаемым в Linux файловым системам относятся:

- *Ext2* — файловая система, изначально разработанная для систем Linux. Сравнительно простая в реализации. Сейчас используется в основном во встраиваемых системах, например, в маршрутизаторах, сотовых телефонах, в качестве корневой файловой системы сетевых накопителей бытового уровня и т.п.
- *Ext3* — дальнейшее развитие *Ext2*, файловая система с поддержкой журналирования. Совместима с *Ext2*. При хранении большого числа файлов в каталогах использование *Ext3* неэффективно.

Лабораторная работа № 1

Знакомство с операционными системами семейства *nix на примере ОС ALT Linux Server.

Командный интерпретатор и основы программирования на shell

Основы регулярных выражений

В отличие от систем семейства Microsoft DOS/Windows, для доступа к файловым системам, расположенным на других дисках, не используются названия этих дисков. Вместо этого файловые системы этих дисков монтируются (mount), или, иными словами, «прикрепляются» к одному из каталогов файловой системы. После монтирования файловая система смонтированного диска становится продолжением общего дерева каталогов в системе. Для прикладных программ не важно, на каком конкретно диске и типе файловой системы находится тот или иной файл — всеми подробностями организации дисков занимается операционная система. Это позволяет скрыть особенности организации дисковой системы от пользователя, легко добавлять и удалять диски из системы, переносить части существующих файловых систем на новые диски без изменения путей к файлам, или, например, разместить часть каталогов файловой системы не на локальных, а на сетевых дисках.

При монтировании можно указывать дополнительные параметры, влияющие на поведение смонтированной файловой системы. В частности, файловые системы могут быть смонтированы в режиме «только для чтения» – в этом случае изменения на них файлов и каталогов будут невозможны. Такой режим монтирования широко используется в операционных системах мобильных устройств, где монтирование в режиме «только для чтения» файловых систем с системными файлами обеспечивает дополнительную защиту от нежелательной модификации служебных данных пользователями. При необходимости обновления системных файлов в этом случае программа-установщик изменяет параметры монтирования файловых систем, делая их доступными для записи, вносит в системные файлы нужные изменения, и возвращает файловые системы обратно в режим доступа только на чтение.

Общая структура каталогов *nix-систем относительно стандартна, современные системы стараются придерживаться рекомендациям *FHS* (*Filesystem Hierarchy Standard*). Рассмотрим общую структуру каталогов *nix-систем на примере ALT Linux Branch 5.1. В корневом каталоге системы располагаются следующие каталоги верхнего уровня:

```
$ ls -1 /
bin
boot
dev
etc
home
lib
lib64
lost+found
media
mnt
opt
proc
root
sbin
srv
```


- *Ext4* — дальнейшее развитие *Ext3*, файловая система с поддержкой журналирования. Совместима с *Ext2* и *Ext3*. Более эффективна при работе с большим числом файлов в каталогах.
- *XFS* — журналируемая файловая система, разработанная для рабочих станций Silicon Graphics (SGI) с операционной системой IRIX. Изначально спроектирована для работы с мультимедийными файлами большого размера, эффективна при использовании расширенных списков контроля доступа к файлам. При использовании *XFS* крайне желательна надёжно работающая аппаратная платформа и наличие резервирования электропитания оборудования.
- *JFS* (*Journaled File System*) — журналируемая файловая система, изначально разработанная корпорацией IBM для ОС AIX.
- *ReiserFS* — журналируемая система, разработанная Гансом Рейзером (Hans Reiser). Оптимизирована для работы с каталогами, содержащими большое количество файлов, а также для хранения небольших по размеру файлов.
- *Btrfs* — журналируемая файловая система, разрабатываемая как замена файловых систем Ext3/Ext4. Обеспечивает эффективную работу с файлами небольшого размера, каталогами с большим числом файлов, имеет возможность прозрачного сжатия хранящихся данных. Поддерживает создание снимков состояния файловой системы, возможность размещения файловой системы на нескольких физических устройствах, оптимизирована под работу с твердотельными дисками (SSD). В настоящее время считается экспериментальной и для широкого использования не рекомендована.
- *ISOFS* (*iso9660*) — файловая система, разработанная для дисков CD, но достаточно часто встречающаяся и на дисках DVD. Имеет ограничение максимального размера файла в 2 Gb.
- *UDF* — файловая система, обычно используемая для дисков DVD.
- *VFAT* — развитие файловой системы MS DOS с добавленной поддержкой длинных имён файлов. Из достоинств файловой системы — простота её реализации. Используется в основном на съёмных носителях данных типа USB Flash. Для разделов дисков, больших 32 Gb, использование *VFAT* крайне неэффективно.
- *NTFS* — файловая система, используемая в ОС Microsoft Windows NT и более поздних. В отличие от *VFAT*, использует журналирование и имеет систему контроля прав доступа к файлам. Использование данной файловой системы в Linux ограничено из-за отсутствия открытой документации по архитектуре файловой системы и сильной зависимости её реализации от архитектуры ОС MS Windows.

Существуют также и специальные файловые системы, из которых можно отметить:

- *procfs* — файловая система, позволяющая обращаться к ряду струк-

например:

- `/var/spool/mail/` — входящие почтовые ящики пользователей;
- `/var/spool/cups/` — очереди документов для печати системы *CUPS* (*Common Unix Printing System*);

- `/var/run/` — информация о запущенных и работающих неинтерактивных программах;
- `/var/tmp/` — различные временные файлы. В отличие от `/tmp/`, содержимое этого каталога должно сохраняться между перезагрузками системы.

Пользователи и процессы в системе.

ОС UNIX изначально разрабатывалась как многопользовательская многозадачная система, и предусматривает одновременную работу многих программ разных пользователей. Каждому пользователю в системе соответствует уникальный числовой идентификатор — *UID* (*User ID*). Хотя для работы самой операционной системы достаточно иметь только числовой идентификатор UID, в целях удобства используются и символьные имена пользователей. Имя пользователя может содержать только латинские строчные буквы, цифры и символ - (дефис). Желательно, чтобы имя пользователя было не длиннее 8 символов, хотя допускаются и более длинные имена. Записи о соответствии символьных имён пользователей их числовым идентификаторам, а также дополнительная информация об учётных записях пользователей хранится в системных информационных базах, простейшим и наиболее распространённым на настольных системах вариантом которых является текстовый файл `/etc/passwd`.

Пользователи объединяются в группы, которые также имеют уникальные числовые идентификаторы — *GID* (*Group ID*), и символьные имена. Записи о группах – соответствие символьных имён числовым идентификаторам, а также списки входящих в группу пользователей, хранятся в своих информационных базах. В простейшем случае настольных систем такая база хранится в файле `/etc/group`. Каждый пользователь должен входить хотя бы в одну группу, которая носит название первичной группы. Также пользователь может входить в одну или несколько других групп, носящих название вторичных. Имя первичной группы пользователя указывается в его записи в файле `/etc/passwd`. Для вторичных групп в `/etc/group` указываются имена пользователей, входящих в них. В системе всегда существует пользователь с *UID*=0 и соответствующая группа с *GID*=0. Этот пользователь является администратором или суперпользователем системы. Традиционно имя суперпользователя — `root`. Принципиальным отличием суперпользователя от остальных пользователей является то, что система не применяет к нему правила контроля доступа, т.е. пользователь с *UID*=0 (`root`) имеет полный и неограниченный доступ ко всех функциям, файлам, устройствами и ресурсам системы.


```
sys
tmp
usr
var
```

- `/bin/` — каталог с основными программами и утилитами. Расположенные в этом каталоге программы жизненно необходимы для функционирования операционной системы и её нормальной загрузки;
- `/boot/` — каталог с файлами ядра операционной системы;
- `/dev/` — каталог с файлами устройств. В старых *nix-системах в каталоге `/dev/` размещались файлы устройств для практически всего поддерживаемого системами оборудования, и размер этого каталога был достаточно большим. В современных дистрибутивах Linux в каталог `/dev/` монтируется файловая система *udevfs*, и в нём присутствуют только файлы реально подключенных и используемых в системе устройств;
- `/etc/` — каталог с файлами конфигурации системы и программ. Практически все настройки системы хранятся в текстовых файлах, которые можно легко просмотреть и изменить обычным текстовым редактором. Как правило, помимо самих настроек в файлах конфигурации размещаются комментарии и описания этих настроек. Обычно комментарии начинаются с символов `#` (октоторп) или `;` (точка с запятой);
  - `/etc/X11/` — каталог с конфигурацией *X Window System, version 11*;
  - `/etc/opt/` — каталог с конфигурацией программ из `/opt/`;
- `/home/` — каталог для домашних каталогов пользователей. Каждому пользователю системы выделяется т.н. домашний каталог — каталог, в котором хранятся личные файлы пользователя, персональные настройки программ и т.п. Например, если в системе есть пользователь `student`, то его домашний каталог будет находится в `/home/student/` ;
- `/lib/` — основные библиотеки системы — т.е. библиотеки, использующиеся программами из каталогов `/bin/` и `/sbin/`. В каталоге `/lib/modules/` размещаются загружаемые модули ядра операционной системы;
- `/lib64/` — каталог для системных библиотек, с указанием архитектуры системы. В данном случае — это каталог для 64-битных библиотек 64-разрядных процессоров с архитектурой Intel. Такие каталоги не обязательны и в ряде систем могут отсутствовать;
- `/lost+found/` — каталоги с таким названием могут присутствовать в корне разделов файловых систем. В ходе проверок файловых систем на целостность после сбоев (например, после отключения питания работающей системы), в эти каталоги помещаются обнаруженные «потерянные» файлы. В нормально работающих системах должны быть пустыми; для некоторых файловых систем (*Ext2*, *Ext3*, *Ext4*) автоматически созда-

запись для каталога — возможность создавать новые файлы или удалять файлы из этого каталога. Наконец, право на исполнение позволяет пользователю запускать файл как программу или сценарий командной оболочки (разумеется, это действие имеет смысл лишь в том случае, если файл является программой или сценарием). Для каталогов право на исполнение имеет особый смысл — оно позволяет сделать данный каталог текущим, т.е. «перейти» в него, например, командой `cd`.

Чтобы получить информацию о правах доступа, можно использовать команду `ls` с ключом `-l`. При этом будет выведена подробная информация о файлах и каталогах, в которой будут, среди прочего, отражены права доступа. Рассмотрим несколько примеров:

```
$ ls -l ~
итого 8
drwx------ 2 student student 4096 Фев 19 17:30 Documents
-rw-r--r-- 1 student student    0 Фев 20 08:03 file.txt
drwx------ 2 student student 4096 Фев 19 15:59 tmp

$ ls -l /var
итого 72
drwxr-xr-x  2 root root   4096 Апр 19  2007 adm
drwxr-xr-x  4 root root   4096 Фев 15 08:32 cache
drwxr-xr-x  2 root root   4096 Апр 19  2007 db
dr-xr-xr-x  2 root root   4096 Апр 19  2007 empty
drwxr-xr-x 11 root root   4096 Фев  9 15:29 lib
drwxr-xr-x  2 root root   4096 Апр 19  2007 local
drwxr-xr-x  6 root root   4096 Фев 20 07:32 lock
drwxr-xr-x 14 root root   4096 Фев 20 07:32 log
lrwxrwxrwx  1 root root     10 Фев  5 13:22 mail -> spool/mail
drwxr-xr-x  2 root root   4096 Апр 19  2007 nis
drwxr-x---  2 root nobody 4096 Апр 19  2007 nobody
drwxr-xr-x  2 root root   4096 Апр 19  2007 opt
drwxr-xr-x  2 root root   4096 Апр 19  2007 preserve
drwxr-xr-x  5 root root   4096 Апр 19  2007 resolv
drwxr-xr-x  5 root root   4096 Фев 17 03:38 run
drwxr-xr-x  6 root root   4096 Фев 20 07:32 spool
drwxrwxrwt  2 root root   4096 Апр 19  2007 tmp
drwxr-xr-x  3 root root   4096 Фев 15 09:24 www
drwx------  2 root root   4096 Апр 19  2007 yp
$ ls -l /bin/su
-rws--x--- 1 root wheel 23712 Окт 18  2006 /bin/su
```

Для файла `~/file.txt` первое поле в строке (`-rw-r--r--`) отражает права доступа. Третье поле указывает на владельца файла (`student`), четвёртое поле указывает на группу, которая владеет этим файлом (`student`). Последнее поле — это имя файла (`file.txt`). Другие поля описаны в документации к команде `ls`.

Данный файл является собственностью пользователя `student` и группы `student`. Последовательность `-rw-r--r--` показывает права доступа для пользователя — владельца файла, пользователей — членов группы-владельца, а также для всех остальных пользователей.

Первый символ из этого ряда обозначает тип файла. Символ – (дефис) означает, что это — обычный файл, который не является каталогом (в этом

Выполняющиеся в системе программы носят названия процессов. Каждый процесс имеет уникальный номер — идентификатор процесса (*PID, Process ID*), а также идентификаторы *UID* и *GID*, с правами которых он выполняется. Любой процесс может с помощью системного вызова `fork()` создать новый процесс. Новый процесс наследует от своего родителя значения *UID* и *GID*. Также процессам доступен системный вызов `chuser()`, который меняет *UID* выполняющего процесса. Вызов `chuser()` доступен только процессам с *UID*=0, т.е. запущенный с правами `root` процесс может один раз изменить свои полномочия на полномочия непривилегированного пользователя, а дальше и этот процесс, и все создаваемые им дочерние процессы изменить свои *UID* не могут. Имеется аналогичный системный вызов и для смены *GID*.

Первый процесс, запускаемый ядром операционной системы при загрузке системы, получает *PID*, равный 1, и выполняется с *UID*=0 и *GID*=0. Обычно этой программой является `/sbin/init`, которая, в свою очередь, запускает другие программы согласно настройкам в `/etc` . Процесс `init` постоянно находится в системе, вплоть до завершения работы.

Все процессы, работающие в системе, можно разделить на три группы. Во-первых, это системные процессы, которые, как и `init`, запускаются ядром. Эти процессы отвечают за работу таких подсистем ядра, как кэширование дисков, управление виртуальной памятью и т.п. Эти процессы запускаются и контролируются непосредственно ядром операционной системы, возможности управления ими весьма ограничены.

Вторая группа — это процессы неинтерактивных системных программ — различных сервисов, выполняющихся в системе. В качестве примера можно привести веб-серверы, серверы баз данных, серверы удалённого доступа к системе, и т.п. Непосредственно с пользователем эти программы не взаимодействуют, для работы с ними требуются дополнительные прикладные программы. Такие процессы, как правило, автоматически запускаются системой при её загрузке, и далее постоянно выполняются в фоновом режиме. Но для функционирования системы они не требуются, и имеется возможность управлять их выполнением — останавливать, запускать и т.п.

К третьей группе относятся прикладные процессы — т.е. программы, непосредственно запускаемые пользователем при его работе с системой.

Кроме суперпользователя и обычных пользователей в системе существует набор т.н. псевдопользователей — непривилегированных пользователей, с правами которых работают различные системные программы. Как правило, псевдопользователям не назначен командный интерпретатор, а их домашний каталог — это тот каталог, в который соответствующие программы могут писать свои данные. При этом в файле `/etc/passwd` вместо командного интерпретатора указывается пустое устройство `/dev/null`.

Системные программы обычно запускаются с привилегиями суперпользователя и после инициализации изменяют с помощью системного вызова

следует выполнить, производят разбор полученных строк, запускают необходимые программы и передают пользователю их вывод — также строки текста. Всё взаимодействие пользователя с системой происходит через командный интерпретатор, поэтому его часто называют оболочкой (*shell*). Последовательности команд для выполнения типовых действий оказываются одинаковыми. Такие последовательности команд можно записать в текстовый файл и далее передать этот текстовый файл командному интерпретатору для выполнения. Такие текстовые файлы называются скриптами. Для запуска они должны иметь соответствующие права (флаг `x`). Командные интерпретаторы поддерживают условное выполнение команд (структуры *if-then-else*), циклы, создание и вызовы подпрограмм и т.п. Язык командного интерпретатора исключительно мощный, и позволяет автоматизировать практически любую задачу в системе. Например, действия при загрузке системы осуществляются скриптами командного интерпретатора — при запуске системы выполняется скрипт `/etc/rc.d/rc.sysinit`, который, в свою очередь, вызывает большое количество других скриптов.

В системах *nix, в соответствии с их модульным построением, доступны несколько командных интерпретаторов. В основном сейчас используется интерпретатор `bash` (`/bin/bash`).

Команды операционной системы представляют из себя небольшие программы, расположенные в каталогах `/bin`, `/usr/bin`, `/sbin`, `/usr/sbin`. В дальнейшем, говоря о командах, мы будем понимать под этим именно указанные программы.

Общий формат вызова команды выглядит следующим образом:

```
$ command -f --flag --key=parameter argument1 agrument2 ...
```

Здесь `$` (знак доллара) — это приглашение операционной системы к вводу команды. Для обычных пользователей оно имеет вид `$`, для суперпользователя (`root`) — # (октоторп). В дальнейшем для команд, которые требуют привилегий `root`, будет использоваться запись вида `# command`.

`command` — имя команды. Для часто использующихся команд имена, как правило, короткие, состоящие из 2-3 букв.

После имени команды, при необходимости, указываются ключи. Ключ — параметр команды, который влияет на результат её выполнения. Часто использующиеся ключи — короткие, односимвольные; для требующихся реже длинных ключей используются слова или сокращения. Короткие ключи начинаются с символа - (дефис), длинные — с двух символов - (дефис). Короткие ключи часто дублируются длинными — для повышения удобства чтения и самодокументирования скриптов. После ключей может допускаться указание дополнительных параметров, для длинных ключей такие пара-

случае первым символом было бы `d`), символьной ссылкой (было бы `l`) или псевдофайлом устройства (было бы `c` или `b`). Следующие три символа (`rw-`) представляют собой права доступа, предоставленные владельцу `student`. Символ `r` — сокращение от read (*англ.* читать), а `w` — сокращение от write (*англ.* писать). Таким образом, `student` имеет право на чтение и запись (изменение) файла `file.txt`.

После символа `w` мог бы стоять символ `x`, означающий наличие прав на исполнение (*англ.* execute, исполнять) файла. Однако символ `-` (дефис), стоящий здесь вместо `x`, указывает, что `student` не имеет права на исполнение этого файла. Это разумно, так как файл `file.txt` не является программой. В то же время, пользователь, зарегистрировавшийся в системе как `student`, при желании может предоставить себе право на исполнение данного файла, поскольку является его владельцем. Для изменения прав доступа к файлу или каталогу используется команда `chmod`.

Следующие три символа (`r--`) отражают права доступа группы к файлу. Группой-владельцем файла в нашем примере является группа `student`. Поскольку здесь присутствует только символ `r`, все пользователи из группы `student` могут читать этот файл, но не могут изменять или исполнять его.

Наконец, последние три символа (это опять `r--`) показывают права доступа к этому файлу всех других пользователей, помимо собственника файла и пользователей из группы `student`. Так как здесь указан только символ `r`, эти пользователи тоже могут лишь читать файл.

Для `~/Documents` первое поле содержит `drwx------`. Это каталог (на что указывает первый символ — буква `d`), владелец которого (`student`) может читать содержимое каталога (т. е. получать список содержащихся в нём файлов), писать в каталог (т. е. изменять его содержимое — создавать, удалять и переименовывать файлы) и переходить в него (для каталогов операцией выполнения считается возможность сделать данный каталог текущим). Другие пользователи — как члены группы `student`, так и все прочие — никаких прав не имеют и ни перейти, ни прочитать содержимое этого каталога, ни, тем более, что-либо в него записать не могут.

Для `/var/adm` первое поле содержит `drwxr-xr-x`. Это каталог, владелец которого — `root` — имеет права `rwx` — т.е. может читать, писать и переходить в этот каталог. Пользователи из группы `root` имеют права `r-x` — т.е. могут читать содержимое каталога и переходить в него. Те же права и у всех остальных пользователей.

Для `/var/empty` первое поле содержит `dr-xr-xr-x`. Это каталог, владелец которого (`root`), группа (`root`) и все остальные пользователи имеют одинаковые права `r-x` — т.е. могут читать содержимое каталога и переходить в него, что соответствует названию каталога (*англ.* empty — пустой). Правда, стоит отметить, что `root` записать что-либо в этот каталог всё-таки может, поскольку на суперпользователя права доступа не распространяются.

`$ rm -f *`	Удалить все файлы в текущем каталоге, не запрашивая разрешений.
`$ rm -r directory/`	Рекурсивно удалить все файлы в каталоге `directory/` и сам каталог `directory/`.
`$ rm -rf *`	Рекурсивно удалить все файлы и каталоги из текущего каталога, не запрашивая подтверждения. Данная команда, отданная от имени суперпользователя и в корневом каталоге, удалит всю файловую систему — **без дополнительных вопросов и возможности отмены действия** (в ряде дистрибутивов в `rm` специально внесены изменения, запрещающие такое поведение).

Для смены даты последнего изменения файла на текущую используется команда `touch <имя файла>`.Если файла не существует, `touch` создаст новый файл нулевого размера.

Для вывода на экран содержимого текстового файла или его части используются команды `cat`, `less`, `more`, `head`, `tail`.

`$ cat /etc/passwd`	Вывести на экран содержимое файла `/etc/passwd`.
`$ more /etc/passwd`	Вывести на экран содержимое файла `/etc/passwd`. Если вывод не будет помещаться на одном экране — вывести начало файла и ждать нажатия любой клавиши для следующей страницы.
`$ less /etc/passwd`	Вывести на экран содержимое файла `/etc/passwd`. Если вывод не будет помещаться на одном экране — вывести начало файла и позволить пользователю просмотреть его, используя прокрутку клавишами управления курсором. Для завершения работы команды `less` следует нажать клавишу `<q>`.
`$ head /etc/passwd`	Вывести первые 10 строк файла `/etc/passwd`.
`$ head -5 /etc/passwd`	Вывести первые 5 строк файла `/etc/passwd`.
`$ tail /etc/passwd`	Вывести последние 10 строк файла `/etc/passwd`.

метры принято записывать через знак `=` (равно). Несколько односимвольных ключей разрешается объединять вместе: например, вместо

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

можно записать:

```
$ ls -la
```

Порядок ключей, как правило, не важен.

После всех ключей следуют аргументы команды. Аргументы чаще всего представляют из себя пути к файлам или каталогам. При необходимости использовать аргументы, начинающиеся со знака `-` (дефис), от списка ключей они отделяются двумя символами `-` (дефис):

```
$ touch -- -file-with-
```

Команды могут использовать различные ключи и параметры. Запоминать все возможные комбинации формата вызова каждой программы невозможно и бессмысленно. Поэтому в системе доступны описания и подсказки по использованию практически каждой утилиты и программы.

Обычно программы поддерживают несколько стандартных ключей. По ключу `-h` или `--help` выдаётся краткая справка о программе. По ключу `-v` или `--version` — её версия. Если краткой справки недостаточно, то можно вызвать описание программы в справочной системе. Для работы со справкой используется команда `man` (сокращение от manual — *англ.* руководство). Команда `man` в качестве аргумента принимает имя команды или файла конфигурации, ищет и выводит на экран страницу справочного руководства. В справке, выдаваемой командой `man`, содержится информация о формате вызова программы, поддерживаемых ей ключах и параметрах, информация об авторах и лицензии программы, в ряде случаев — примеры использования, ссылки на сайты разработчиков с дополнительной документацией.

Для просмотра страниц руководства, не помещающихся на экране, следует использовать прокрутку клавишами перемещения курсором, `<Page Up>` и `<Page Down>`. Пробел перемещает руководство на страницу вперёд. Для выхода из `man` и продолжения работы с системой следует нажать клавишу `<q>` (от *англ.* quit, выйти).

Часть программ, помимо руководств в формате `man`, также имеют и более пространную документацию в формате `info` — с вызовом её через одноимённую утилиту.

В отличие от встроенной системы подсказки программ в операционной системе Windows, руководства `man` и `info` содержат полную подробную техническую информацию о работе команд.

`# su -l user`	Запустить командный интерпретатор с правами пользователя `user`.
`# useradd user`	Добавить учётную запись пользователя `user`. При этом создаются необходимые записи в файлах `/etc/passwd` и `/etc/group`, а также домашний каталог пользователя. Пароль новому пользователю не назначается, и войти в систему до его задания он не может.
`# userdel user`	Удалить учётную запись пользователя `user`. Файлы, принадлежащие пользователю, при этом не удаляются.
`# passwd user`	Задать пароль пользователя `user`. Суперпользователю знать старый пароль `user` для его смены не нужно.
`# chmod <права> file`	Изменить права на файл. `root` может изменить права доступа к любому файлу.
`# chown user file`	Изменить владельца файла на пользователя `user`.
`# chown :group file`	Изменить группу файла на `group`.
`# chown user:group *`	Изменить владельца и группу всех файлов в текущем каталоге.
`# chown -R user:group *`	Рекурсивно изменить владельца и группу всех файлов в текущем каталоге и подкаталогах.
`# shutdown`	Выключить систему. Без дополнительных ключей команда `shutdown` останавливает систему, не отключая её питание. Удалённо выключить систему можно, но включить её после этого возможно только внешними средствами.
`# halt`	Выключить систему, не отключая её питание (аналогично вызову `shutdown`).
`# shutdown -h`	Выключить систему, и отключить её питание.
`# poweroff`	Выключить систему, в т.ч. отключить её питание (аналогично вызову `shutdown -h`).
`# shutdown -r`	Перезагрузить систему.
`# reboot`	Перезагрузить систему (аналогично вызову `shutdown -r`).

При работе с правами суперпользователя следует помнить, что никаких ограничений и прав доступа для этой учётной записи не существует. Поэтому неосторожная команда или опечатка может привести систему в

`$ tail -1 /etc/passwd`	Вывести последнюю строку файла `/etc/passwd`.

Правами доступа к файлам можно управлять командой `chmod`:

`$ chmod u+rwx file.sh`	Добавить файлу `file.sh` право владельцу на чтение, запись и выполнение.
`$ chmod g+x files.sh`	Добавить для группы файла `files.sh` право на выполнение.
`$ chmod u=rw,g=r,o-rwx file.txt`	Для файла `file.txt`: установить права для владельца в `rw-`, для группы — в `r--`, для всех остальных — в `---`.
`$ chmod a+rx file.sh`	Добавить для `file.sh` права на чтение и выполнение для всех пользователей.
`$ chmod g-w,o-rwx file.txt`	Снять с файла `file.txt` права на запись для группы пользователей и все права для остальных пользователей.
`$ chmod -R g+w directory/`	Добавить для всех файлов и каталогов внутри каталога `directory/` прав на запись для группы.
`$ chmod -R g+X directory/`	Добавить для всех *каталогов* внутри каталога `directory/` право на выполнение, а для *файлов* право на выполнение оставить прежним.

Для смены своего пароля пользователь может использовать команду `passwd`. При запуске команда запросит у пользователя его текущий пароль, новый пароль и для подтверждения ввода — повторение нового пароля. Ввод паролей на экране не отображается. При совпадении введённых паролей, пароль пользователя будет изменён. Важно помнить, что пароль является средством, по которому система аутентифицирует пользователя. Короткие или легко угадывающиеся пароли очень быстро и просто находятся путём их перебора. Учитывая, что *nix — это сетевые операционные системы, слабые пароли пользователей легко позволяют злоумышленникам подбирать их и проникать в системы.

В настоящее время пароль не должен содержать менее 8 символов. Эти символы не должны быть одинаковыми, пароль не должен содержать только цифры или быть словарным словом. Все подобные, слабые пароли легко определяются современными программами для взлома систем в автомати-

буквой 'y':

```
$ yes
y
y
y
y
y
```

Последовательность таких строк будет бесконечно продолжаться – пока выполняется команда `yes` . Остановить её выполнение можно, отправив команде сигнал прерывания, т.е. нажав <Ctrl>+<C>.

Чтобы на экран не выводилась эта бесконечная последовательность, перенаправим стандартный вывод команды `yes` на `/dev/null`. Устройство `/dev/null` — одно из специальных устройств в системе, оно действует как «чёрная дыра»: все данные, посланные в это устройство, пропадают. С помощью этого устройства очень удобно избавляться от слишком обильного вывода некоторых программ. Подробнее о перенаправлении устройств ввода-вывода рассказано ниже по тексту в соответствующем разделе.

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

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

Можно сделать так, чтобы команда `yes` продолжала работать, но при этом приглашение командной оболочки вернулось на экран и стало возможно работать с другими программами. Для этого можно команду `yes` перевести в фоновый режим, и она будет там выполняться параллельно с другими запускаемыми из командного интерпретатора программами.

Один из способов запустить процесс в фоновом режиме — дописать символ `&` (амперсанд) в конце строки запуска команды:

```
$ yes > /dev/null &
[1]+ 164
$
```

Сообщение `[1]` представляет собой номер задания (*англ.* job number) для процесса `yes`. Командная оболочка присваивает номер задания каждому исполняемому заданию. Поскольку `yes` является единственным исполняемым заданием в данном сеансе, ему был присвоен порядковый номер 1. Число 164 является идентификационным номером, соответствующим данному процессу (*PID*); он уникален для системы в целом. К запущенному в фоновом режиме процессу можно обращаться, указывая как его *PID*, так и номер задания.

нерабочее состояние.

Управление выполнением программ.

Каждая выполняющаяся в Linux программа называется процессом. Linux, как многопользовательская многозадачная система характеризуется тем, что в ней одновременно может выполняться множество процессов, принадлежащих разным пользователям. Вывести список исполняющихся в текущее время процессов можно командой `ps`, например, следующим образом:

```
$ ps
PID TT STAT  TIME COMMAND
24  3  S     0:03 bash
161 3  R     0:00 ps
$
```

По-умолчанию команда `ps` выводит список только тех процессов, которые принадлежат запустившему её пользователю и выполняются в данной сессии. Чтобы посмотреть все исполняющиеся в системе процессы, нужно использовать ключ `-a`, т. е. запускать команду как `ps -a`. Наиболее полный вид списка процессов, с указанием их владельцев, времени запуска, потребляемых ресурсов (памяти и процессора) можно просмотреть командой `ps -aux`.

Номера процессов (*process ID*, или *PID*), указанные в первой колонке, являются уникальными номерами, которые система присваивает каждому работающему процессу. Последняя колонка, озаглавленная `COMMAND`, показывает имя работающей команды. Среди команд, запущенных данным пользователем, есть только `bash` и сама команда `ps`. (`bash` — это командный интерпретатор (командная оболочка, *англ.* shell), который обрабатывает вводимые пользователем с терминала команды и обеспечивает их выполнение в системе. Более подробно роль командного интерпретатора рассматривалась в предыдущей лабораторной работе.) Видно, что командная оболочка `bash` выполняется одновременно с командой `ps`. Когда пользователь ввёл команду `ps`, оболочка `bash` начала её исполнять. После того, как команда `ps` закончила свою работу (таблица процессов выведена на экран), управление возвращается процессу `bash`. Тогда оболочка `bash` выводит на экран приглашение и ждёт новой команды.

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

В большинстве случаев пользователи в каждый момент времени запускают только одно задание — ту команду, которую они ввели и запустили из командной оболочки. Однако многие командные оболочки (включая `bash` и

```
0
$ ls /tmp/0
ls: невозможно получить доступ к /tmp/0: Нет такого файла или каталога
$ echo $?
2
```

Здесь сначала успешно выводится список файлов из (пустого) каталога `/tmp`, а далее при попытке обратиться к несуществующему `/tmp/0` возникает ошибка. При этом `ls` как выводит сообщение об ошибке, так и возвращает ненулевой код возврата, сигнализирующий о ней.

Управление последовательностью выполнения команд.

В строке ввода интерпретатор команд позволяет ввести и запустить сразу несколько разных команд. Если команды требуется просто запускать последовательно одну за другой без учёта результата выполнения предыдущей команды перед запуском следующей, то их достаточно разделить точкой с запятой:

```
$ cd /bin; ls -l sh
-rwxr-xr-x 1 root root 486600 апр 19  2013 sh
```

Но также при запуске последующей команды можно и учитывать результат выполнения предыдущей. Если команда завершилась успешно (т. е. её код возврата равен нулю), то командный интерпретатор считает, что результат выполнения команды — логическая истина. Если код возврата отличен от нуля (т. е. произошла какая-либо ошибка), то результат выполнения команды – логическая ложь.

Для запуска следующей команды только в том случае, если предыдущая команда завершилась успешно, используется оператор «логическое И», записываемый как `&&` :

```
$ cd /tmp/ && touch file
```

Здесь команда `touch file` запускается только после успешного выполнения команды `cd /tmp`, т. е. после перехода в каталог `/tmp/` . В случае невозможности перехода в каталог команда `touch` запущена не будет.

Для запуска следующей команды только в том случае, если предыдущая завершилась с ошибкой, используется оператор «логическое ИЛИ», записываемый как `||` :

```
$ cd /tmp/0 || mkdir /tmp/0
```

Здесь делается попытка перехода в каталог `/tmp/0`, и если это не удаётся (например, такого каталога нет), запускается команда `mkdir /tmp/0` , создающая этот каталог.

Использование операторов «логического И» и «ИЛИ» для условий выполнения команды в зависимости от результата предыдущей команды основывается на логике оптимизации выполнения этих операций в языках программирования: результатом «логического И» будет логическая истина

Для того, чтобы проверить состояние запущенного и работающего в фоновом режиме процесса, можно использовать команду `jobs`, которая является внутренней командой оболочки.

```
$ jobs
[1]+  Running                 yes >/dev/null  &
$
```

В выводе команды `jobs` указывается, какие задания запущены, их номера, текущее состояние (выполняется, приостановлено, ожидает ввода-вывода) и вид командной строки. Также для того, чтобы узнать статус задания, можно воспользоваться командой `ps`, как это было показано выше.

Для того, чтобы передать процессу сигнал (чаще всего когда возникает потребность прервать работу задания) используется упомянутая выше утилита `kill`. В качестве аргумента этой команде даётся либо номер задания, либо *PID*. Необязательный параметр — номер сигнала, который нужно отправить процессу. По умолчанию отправляется сигнал `TERM`. Если к заданию нужно обратиться по его номеру (а не через *PID*), то номер задания в параметрах команды `kill` указывается через символ `%` (процент). В рассмотренном выше случае номер задания был 1, так что команда `kill %1` прервёт работу задания:

```
$ kill %1
$ jobs [1]
Terminated              yes       >/dev/null
```

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

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

Приостановка и продолжение работы заданий.

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

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

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

Теперь вместо того, чтобы прервать задание комбинацией клавиш `<Ctrl>+<C>`, приостановим его (suspend, *англ.* подвесить), отправив сигнал `STOP`. Для приостановки задания надо нажать соответствующую комбина-

Если вывод команды не интересен, его можно перенаправить на специальное устройство `/dev/null` — как говорилось выше, все данные, посланные в это устройство, удаляются. Также существуют специальные устройства `/dev/zero` – из которого можно прочитать неограниченное число нулевых символов, `/dev/random` – из которого можно прочитать случайные символы, `/dev/urandom` – для чтения последовательности псевдослучайных символов.

Использование состыкованных команд (конвейер).

Выше уже демонстрировалось, как использовать программу `sort` в качестве фильтра. В этих примерах предполагалось, что исходные данные находятся в некотором файле, или что эти исходные данные будут введены с клавиатуры (стандартного ввода). Однако часто требуется отсортировать данные, которые являются результатом работы какой-либо другой команды, например, `ls` .

Будем сортировать данные в обратном алфавитном порядке, это делается опцией `-r` команды `sort`. Если нужно перечислить файлы в текущем каталоге в обратном алфавитном порядке, один из способов сделать это будет следующим. Для получения списка файлов используем команду `ls` :

```
$ ls /bin
arch
awk
basename
bash
....
$
```

Теперь перенаправляем выход команды `ls` в файл с именем `file-list`, и далее сортируем этот файл с помощью команды `sort`:

```
$ ls /bin > file-list
$ sort -r file-list
zcat
ypdomainname
xargs
wc
...
$
```

Здесь вывод команды `ls` был сохранён в файле, а после этого файл был обработан командой `sort -r`. Однако этот путь является неэффективным — он требует использования временного файла для хранения выходных данных программы `ls` , лишних операций ввода-вывода для создания, записи и последующего чтения этого временного файла с диска.

Решением в данной ситуации может служить создание состыкованных команд (*англ.* pipelines). Стыковку осуществляет командная оболочка, которая *stdout* первой команды направляет на *stdin* второй команды. В данном случае мы хотим направить *stdout* команды `ls` на *stdin* команды `sort`. Для

в случае, если оба операнда равны логической истине. Если первый операнд – логическая ложь, то результат – логическая ложь при любом значении второго операнда, и его можно не вычислять. Аналогично, результатом «логического ИЛИ» будет логическая истина в случае, если один из операндов равен логической истине. Соответственно, если первый операнд равен логической истине, то результат уже известен, и значение второго операнда вычислять смысла нет.

Потоки ввода-вывода и их перенаправление.

Программы нужны для того, чтобы обрабатывать данные: принимать одно, на выходе выдавать другое, причём в качестве данных может выступать практически что угодно: текст, числа, звук, видео и т.д. Потоки входных и выходных данных для команды называются вводом и выводом. Потоков ввода и вывода у каждой программы может быть и по несколько. В Linux каждый процесс при создании в обязательном порядке получает так называемые стандартный ввод (*англ.* standard input, *stdin*), стандартный вывод (*англ.* standard output, *stdout*) и стандартный вывод ошибок (*англ.* standard error, *stderr*).

Программы работают с потоками ввода-вывода как с обычными файлами. С точки зрения программирования потоки ввода-вывода — это доступные сразу после запуска программы заранее открытые файловые дескрипторы с номерами 0, 1 и 2 для стандартного ввода, стандартного вывода и стандартного вывода ошибок соответственно. При необходимости программы могут переопределять эти файловые дескрипторы, закрывать их, и т.д.

Стандартные потоки ввода/вывода предназначены в первую очередь для обмена текстовой информацией. Тут даже не важно, кто общается с помощью текстов, человек с программой или программы между собой —главное, чтобы у них был канал передачи данных, и чтобы они говорили «на одном языке».

Текстовый принцип работы с машиной позволяет отвлечься от конкретных частей компьютера, вроде системной клавиатуры и видеокарты с монитором, рассматривая единое оконечное устройство, посредством которого пользователь вводит текст (команды) и передаёт его системе, а система выводит необходимые пользователю данные и сообщения (диагностику и ошибки). Такое устройство называется терминалом. В общем случае терминал — это точка входа пользователя в систему, обладающая способностью передавать текстовую информацию. Терминалом может быть отдельное внешнее устройство, подключаемое к компьютеру через порт последовательной передачи данных (*COM port* в терминологии персональных компьютеров). В роли терминала также могут работать и специальные программы: например, *PyTTY* и серверная часть — демон удалённого управления системой `ssh`. При работе с командной строкой стандартный ввод командной оболочки связан с клавиатурой, а стандартный вывод и вывод ошибок — с экраном монитора (или окном эмулятора терминала).

Рассмотрим в качестве примера одну из простейших команд — `cat`.

Одновременное перенаправление в один и тот же файл и потока стандартного вывода, и потока ошибок встречается очень часто — для упрощения записи в ряде командных интерпретаторов, в т.ч. в Bash, есть дополнительный оператор перенаправления `&>` , переназначающий оба потока вывода сразу:

```
$ mkdir /etc/my-directory &> /dev/null
```

Узнать о результате выполнения команды при перенаправлении всего её вывода в устройство `/dev/null` можно, проанализировав код возврата.

Основы регулярных выражений.

Регулярные выражения (*англ.* regular expressions, сокращённо *regex*) — это система поиска фрагментов в тексте, основанная на специальной системе записи образцов для поиска. Образец (*англ.* pattern), задающий правило поиска, также называют шаблоном или маской.

Сейчас регулярные выражения используются многими текстовыми редакторами и утилитами для поиска и изменения текста на основе выбранных правил. Многие языки программирования имеют встроенную поддержку работы с регулярными выражениями, для других они доступны как внешние библиотеки. Набор утилит (включая редактор `sed` и фильтр `grep`), поставляемых в дистрибутивах *nix, одним из первых способствовал распространению регулярных выражений.

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:

- Перечисление: вертикальная черта разделяет допустимые варианты. Например, «`one|two`» соответствует *one* или *two*.
- Группировка: круглые скобки используются для определения области действия и приоритета операторов. Например, шаблоны «`abd|acd`» и «`a(b|c)d`» описывают одно и то же множество: *abd* и *acd*.
- Квантификация: квантификатор после символа или группы символов определяет, сколько раз предшествующее выражение может встречаться. Например:
  - `{m,n}` — общее выражение, повторений может быть от `m` до `n` включительно.
  - `{m,}` — общее выражение, `m` и более повторений.
  - `{,n}` —общее выражение, не более `n` повторений.
  - `?` (вопросительный знак) означает `0` или `1` раз, то же самое, что и `{0,1}`. Например, «`colou?r`» соответствует и *color*, и *colour*.

стыковки используется символ `|` (вертикальная черта), как это показано в следующем примере:

```
$ ls /bin | sort -r
zcat
ypdomainname
xargs
wc
...
$
```

Эта команда короче, чем последовательность отдельных команд, и её проще набирать.

Рассмотрим ещё один пример. Команда

```
$ ls /usr/bin
```

выдаёт длинный список файлов. Большая часть этого списка выводится на экран слишком быстро, чтобы его содержимое можно было прочитать. Попробуем использовать команду `more` для того, чтобы выводить этот список частями:

```
$ ls /usr/bin | more
```

Теперь можно этот список «перелистывать».

Можно пойти дальше и состыковать более двух команд. Рассмотрим команду `head`, которая является фильтром, выводящим первые строки из входного потока (в нашем случае на вход будет подан выход от нескольких состыкованных команд). Если мы хотим вывести на экран последнее по алфавиту имя файла в текущем каталоге, можно использовать следующую длинную команду:

```
$ ls | sort -r | head -1 notes
```

где команда `head -1` выводит на экран первую строку получаемого ей входного потока строк (в нашем случае поток состоит из данных от команды `ls`), отсортированных в обратном алфавитном порядке.

Фильтры не обязательно используются только для обработки текста. Например, в пакете `netpbm` содержатся утилиты для обработки изображений, которые тоже являются фильтрами. Для увеличения иконки *Midnight Commander* в 5 раз и преобразования её из формата *PNG* в *JPEG* можно использовать такую связку команд:

```
$ pngtopnm /usr/share/icons/mc.png | pnmenlarge 5 | pnmsmooth | pnmtojpeg >
/tmp/mc.jpg
```

Здесь `pngtopnm` читает файл иконки (`/usr/share/icons/mc.png`) в формате *PNG*, преобразует его в формат *PNM* и выдаёт результат в стандартный вывод. `pnmenlarge` принимает файл в формате *PNM* из стандартного ввода, увеличивает (масштабирует) картинку в 5 раз и выдаёт результат в стандартный вывод. Далее `pnmsmooth` выполняет операцию сглаживания, а `pnmtojpeg` преобразует поток данных в формат *JPEG*. Итоговый результат

Рассмотрим некоторые примеры использования `grep` и регулярных выражений. Как говорилось в предыдущей лабораторной работе, команда `ls` выводит список файлов в каталоге. Команда `ls /bin` выведет список файлов из каталога `/bin`. Вывод команда `ls` осуществляет в *stdout*.

Предположим, нас интересуют те программы (файлы) из `/bin`, которые содержат подстроку *zip*. Этой подстроке соответствует простейшее регулярное выражение «`zip`». Перенаправляем вывод из `ls` в `grep` и получаем:

```
$ ls /bin | grep 'zip'
bunzip2
bzip2
bzip2recover
gunzip
gzip
```

Здесь регулярное выражение заключено в одиночные кавычки `''`, которые указывают `bash`, что внутри них — обычная строка. Такой синтаксис позволяет использовать в регулярном выражении пробелы, и его разумно придерживаться во всех случаях (например, регулярное выражение `'a b'` описывает шаблон для строк, содержащих последовательно *a*, пробел и *b*. Если этот шаблон указать `grep` без кавычек, т.е. `grep a b`, то командный интерпретатор, разобрав строку, вызовет `grep` с двумя параметрами, и `grep` будет искать строки с буквами *a* в файле `b`. При использовании кавычек командный интерпретатор будет считать выражение `'a b'` одним параметром, и передаст его `grep` целиком, вместе с пробелом внутри).

Файлы из `/bin`, которые кончаются на *2*:

```
$ ls /bin | grep '2$'
bash2
bunzip2
bzip2
```

Файлы из `/bin`, которые начинаются на *b*:

```
$ ls /bin | grep '^b'
basename
bash
bash2
bunzip2
bzcat
bzip2
bzip2recover
```

Файлы из `/bin`, начинающиеся на *b* и содержащие в своём имени букву *a*:

```
$ ls /bin | grep '^b.*a'
basename
bash
bash2
bzcat
```


- `*` (астериск) означает `0`, `1` или любое число раз (`{0,}`). Например, «`go*gle`» соответствует *ggle*, *gogle*, *google* и т.д.
- `+` (плюс) означает хотя бы `1` раз (`{1,}`). Например, «`go+gle`» соответствует *gogle*, *google* и т.д. (но не *ggle*).

Конкретный синтаксис регулярных выражений зависит от их программной реализации. Мы будем рассматривать синтаксис «базовых» регулярных выражений UNIX. Хотя он на данный момент и определён *POSIX* как устаревший, но до сих пор широко распространён из соображений обратной совместимости. Многие UNIX-утилиты используют такие регулярные выражения по умолчанию.

В этом синтаксисе большинство символов соответствуют сами себе («`a`» соответствует *a* и т.д.). Исключения из этого правила называются метасимволами:

`.`	Соответствует любому единичному символу.
`[ ]`	Соответствует любому единичному символу из числа заключённых в скобки. Символ - (дефис) интерпретируется буквально только в том случае, если он расположен непосредственно после открывающей или перед закрывающей скобкой: `[abc-]` или `[-abc]`. В противном случае он обозначает интервал символов. Например, `[abc]` соответствует *a*, *b* или *c*. `[0-9]` соответствует цифрам.
`[^ ]`	Соответствует единичному символу из числа тех, которых нет в скобках. Например, `[^abc]` соответствует любому символу, кроме *a*, *b* или *c*. `[^0-9]` соответствует любому символу, кроме цифр.
`^`	Используемое в начале регулярного выражения, соответствует началу строки текста.
`$`	Используемое в конце регулярного выражения, соответствует концу строки текста.
`\(\)`	Объявляет «отмеченное подвыражение», которое может быть использовано позже.
`\n`	`n` — цифра от `1` до `9`, соответствует *n*-му отмеченному подвыражению.
`*`	Астериск после выражения, соответствующего единичному символу, соответствует нулю или более копий этого выражения. Например, «`[xyz]*`» соответствует пустой строке, *x*, *y*, *zx*, *zyx*, и т.д.
`\{x,y\}`	Соответствует последнему блоку, встречающемуся не менее `x` и не более `y` раз. Например, «`a\{3,5\}`» соответствует *aaa*,

```
$ uptime | sed 's/^.* up \+\(.\+\), \+[0-9]\+ \+user.*/\1/'
27 days, 22:13
```

Здесь мы отметили, что время работы системы начинается за словом *up*, а после него идёт число пользователей. Соответственно, требующееся регулярное выражение для помещения времени работы системы в подстроку можно описать как:

- любое число любых символов от начала строки, далее пробел и слово *up* — `^.* up`
- за которым следует через один или несколько пробелов время работы системы — `^.* up \+\(\)`
- само время работы системы может содержать фактически любые символы, в т.ч. пробелы, знаки пунктуации и пр. — `^.* up \+\(.\+\)`
- однако за ним через запятую и один или несколько пробелов — `^.* up \+\(.\+\), \+`
- следует количество пользователей (число, одна или несколько цифр) — `^.* up \+\(.\+\), \+[0-9]\+`
- и слово *user* (или *users*). Далее до конца строки может быть что угодно — `^.* up \+\(.\+\), \+[0-9]\+ \+user.*`

Отметим, что то же самое мы могли бы сделать и по-другому: просто удаляя из вывода ненужный нам текст. Например:

```
$ uptime | sed 's/user.*//'
 08:18:07 up 27 days, 22:43,  2
```

убирает весь текст от *user* включительно и до конца строки. Также убираем в полученном результате и всё в конце строки от запятой включительно:

```
$ uptime | sed 's/user.*//'| sed 's/,[^,]*$//'
 08:24:13 up 27 days, 22:49
```

Отметим, что более простой вариант без привязки к концу строки

```
$ uptime | sed 's/user.*//'| sed 's/,[^,]*//'
 08:24:18 up 27 days,  2
```

из-за «ленивости» регулярных выражений совпадёт с первым вхождением запятой (*, 22:43*), а ещё более простой вариант

```
$ uptime | sed 's/user.*//'| sed 's/,.*$//'
 08:25:11 up 27 days
```

из-за «жадности» будет совпадать с текстом от первой запятой до конца строки (*, 22:43, 2*).

Далее нам нужно удалить текст от начала строки до *up* включительно:

```
$ uptime | sed 's/user.*//'| sed 's/,[^,]*$//' | sed 's/^.*up \+//'
27 days, 22:54
```

и мы получаем требуемый результат. (Символ `\` (обратный слеш) в конце

Здесь в регулярном выражении указано, что оно:

- должно совпадать с началом строки — `^`
- в начале строки должна быть буква *b* — `^b`
- дальше может быть любой символ — `^b.`
- и таких символов может быть сколько угодно — 0 или больше — `^b.*`
- а дальше должна быть буква *a* — `^b.*a`

Файлы из `/bin`, начинающиеся на *b* и содержащие в своём имени буквы *a*, *e* или *k*:

```
$ ls /bin | grep '^b.*[aek]'
basename
bash
bash2
bzcat
bzip2recover
```

Здесь используется описание набора символов — `[aek]`.

Рассмотрим более полезный пример.

На предыдущей лабораторной работе производилась настройка сервера `lighttpd`. Его конфигурационный файл — `/etc/lighttpd/lighttpd.conf`. Как было видно, в нём (как и в большинстве других конфигурационных файлов) содержится большое количество комментариев, как с поясняющим текстом, так и с примерами различных опций настройки. Предположим, нам нужно посмотреть текущую конфигурацию сервера. Однако посмотреть её простой командой `cat /etc/lighttpd/lighttpd.conf` неудобно: текст не помещается на экране. Мы можем, конечно, использовать команду `less` для прокрутки текста, но комментарии при этом всё равно будут мешать. Мы можем удалить их из файла, но тогда сложно будет что-либо изменять в нём в дальнейшем.

Проще отфильтровать ненужный текст непосредственно при выводе файла на экран.

Комментарии в `lighttpd.conf` начинаются с символа `#` (октоторп). Перед ним в начале строки может или не быть ничего, или быть один или несколько пробелов.

Таким образом, регулярное выражение для выделения строк с комментариями — «`^ *#`»: начало строки, ноль или несколько пробелов, и затем — *#*.

Кроме того, нас не очень интересуют просто пустые строки, в которых нет никакого текста. Такие строки можно описать выражением «`^$`»: начало строки, и сразу — её конец. Может быть и другой вариант: строка, состоящая из одних пробелов, которая также не несёт никакой информации. Таким образом, общее регулярное выражение приобретает вид «`^ *$`».

нестандартного каталога требуется указывать путь к ней, т.е., в данном случае, запустить программу как `script` нельзя — вместо созданного нами скрипта командный интерпретатор запустит стандартную утилиту `script` из `/usr/bin`.

Часто простого последовательного выполнения недостаточно: для эффективного программирования требуются переменные, условное выполнение команд и т.п. Командный интерпретатор имеет собственный язык, который по своим возможностям приближается к высокоуровневым языкам программирования. Этот язык позволяет создавать программы (*shell*-файлы, *shell*-скрипты), которые могут включать операторы языка и команды UNIX.

Такие файлы не требуют компиляции и выполняются в режиме интерпретации, но они, как отмечалось ранее, должны обладать правом на исполнение (устанавливается с помощью команды `chmod`).

Скрипту могут быть переданы аргументы при запуске. Каждому из первых девяти аргументов ставится в соответствие позиционный параметр от `$1` до `$9` (`$0` — имя самого скрипта), и по этим именам к ним можно обращаться из текста скрипта.

Прежде чем начать рассмотрение некоторых операторов *shell*, следует обратить внимание на использование в командах некоторых символов.

- $ (знак доллара) – используется для подстановки в строку значения переменной, имя которой указывается сразу за ним (`$VAR`).
- ``` `` ``` (обратные апострофы) — служат выполнения команды, заключённой между ними, и подстановки в строку вывода этой команды.
- \ (обратный слеш) — знак отмены специального значения («экранирования») следующего за ним символа, такого как $ или `. Будучи последним символом в строке, обратный слэш экранирует символ перевода строки и позволяет разбивать запись команд с многочисленными и длинными аргументами на несколько строк
- "" (двойные кавычки) — используются для обрамления текста, внутри которого командная оболочка выполняет поиск и интерпретацию специальных символов.
- '' (одинарные кавычки или апострофы) — используются для обрамления текста, передаваемого как единый аргумент команды или присваиваемого переменной без интерпретирования в нём специальных символов.

Кроме того, для удобства работы с файлами почти все командные интерпретаторы интерпретируют символы `?` (знак вопроса) и `*` (астериск), используя их как шаблоны имен файлов (т.н. метасимволы):

строки здесь означает, что команда будет продолжена на следующей строке).

Утилита awk.

AWK — интерпретируемый скриптовый язык, предназначенный для обработки текстовой информации. Первая версия *AWK* была написана в 1977 году в AT&T Bell Laboratories и получила название по фамилиям своих разработчиков: Альфреда Ахо (Alfred V. Aho), Питера Вейнбергера (Peter J. Weinberger) и Брайана Кернигана (Brian W. Kernighan).

AWK рассматривает входной поток как набор записей, каждая из которых состоит из набора полей. По умолчанию для *AWK* записью является строка, а разделителями полей в строке — пробелы. Внутри программы на *AWK* значение поля можно получить как значение переменной `$1`, `$2`, `$3`, ... Переменная `$0` содержит в себе всю запись.

Программа на *AWK* имеет вид

```
BEGIN{ДЕЙСТВИЕ}
ШАБЛОН {ДЕЙСТВИЕ}
ШАБЛОН {ДЕЙСТВИЕ}
…
{ДЕЙСТВИЕ}
END{ДЕЙСТВИЕ}
```

Для каждой строки, совпадающей с шаблоном, выполняется указанное действие. Если шаблон не указан, то действие выполняется для всех строк. Опционально можно указать блоки кода `BEGIN{}` и `END{}`, которые будут выполняться один раз, до первой входной строки и после последней входной строки соответственно.

Шаблон — это регулярное выражение, из большого числа возможных действий мы рассмотрим только команду `print`.

Рассмотрим использование команды `awk` на примерах.

Список файлов с указанием их владельцев, прав, и даты последнего изменения можно получить командой `ls -l`. Он имеет вид:

```
$ ls -l /bin | head -n 5
total 5596
lrwxrwxrwx 1 root root       4 Feb 25 05:30 awk -> gawk
-rwxr-xr-x 1 root root   19064 Apr 20  2008 basename
-rwxr-xr-x 1 root root  549368 Mar 27  2008 bash
lrwxrwxrwx 1 root root       4 Feb 25 05:30 bash2 -> bash
```

Преобразуем этот список в формат

```
<имя файла> <владелец>:<группа> <права>
```

`awk` обрабатывает каждую строку списка отдельно, и самостоятельно разбивает её на поля по границам слов. Права файла — поле 1, владелец и группа — поля 3 и 4, имя файла — поле 9. Тогда:

```
$ if test -f /bin/bash; then echo 'bash найден!'; fi
bash найден!
```

можно использовать более аккуратно выглядящую конструкцию

```
$ if [ -f /bin/bash ]; then echo 'bash найден!'; fi
bash найден!
```

Построение циклов.

В языке командного интерпретатора существует три типа циклов: `while`, `until` и `for`.

Цикл `while`:

```
while список_команд1; do
     список_команд2
done
```

В условии учитывается код возврата последней выполненной команды из `списка_команд1`, при этом 0 интерпретируется как «истина».

Цикл `until`:

```
until список_команд1; do
     список_команд2{;|перевод строки}
done
```

Проверка условия выполняется перед выполнением цикла. Учитывается код возврата последней выполненной команды из `списка_команд1`, при этом цикл выполняется до тех пор, пока код возврата не примет значение «истина», т. е. будет равным нулю.

Цикл `for`:

```
for переменная [in список_значений]; do
     список_команд
done
```

Переменной присваивается значение очередного слова из `списка_значений`, и для этого значения выполняется `список_команд`. Количество итераций равно количеству цепочек символов в `списке_значений`, разделённых пробелами. Если ключевое слово `in` и `список_значений` опущены как необязательные, то переменной поочередно присваиваются значения параметров, переданных при запуске программы-скрипта. В качестве передаваемых параметров можно использовать шаблоны имён файлов, тогда интерпретатор превращает эти шаблоны в список имён файлов, удовлетворяющих шаблону.

Например,

```
$ A=1; for i in `ls /bin | grep '^b'`; do
> echo "$A  :$i"
> A=`expr $A + 1`
```


- `?` — один любой символ;
- `*` — произвольное количество любых символов.

Например, `*.c` обозначает все файлы с расширением `c`, `pr???.*` обозначает файлы, имена которых начинаются с `pr`, содержат пять символов и имеют любое расширение.

Переменные языка shell.

Язык *shell* позволяет работать с переменными без предварительного объявления. Имена переменных начинаются с латинской буквы и могут содержать латинские буквы, цифры и символ подчеркивания. Обращение к переменным начинается со знака `$` (знак доллара).

Имеется большое количество уже определённых переменных — т.н. переменных окружения. Их полный список можно получить командой `set`. Переменные окружения используются для настройки различных параметров окружения пользователя, например, в переменной `TMP` задаётся каталог для временных файлов, используемый рядом программ:

```
$ echo $TMP
/tmp/.private/student
$ ls $TMP
mc-student
```

Переопределить (в т.ч. случайно) такие системные переменные можно, но стоит учесть, что это может привести к нежелательным последствиям.

Для разных пользователей могут быть разные наборы переменных окружения с разными значениями. Например, как говорилось ранее, командный интерпретатор ищет выполняемые файлы в определённых каталогах: `/bin`, `/usr/bin` и т.п. Перечень этих каталогов командный интерпретатор берёт из переменной окружения `$PATH` . Для суперпользователя в этой переменной, помимо каталогов с программами пользователя, также указываются каталоги системных программ - `/sbin`, `/usr/sbin`. Или, для обычного пользователя в нескольких переменных окружения с именами вида LC_* задаются настройки локали — с учётом родного языка пользователя. Для суперпользователя используется английская локаль – для избежания проблем с поведением регулярных выражений в системных скриптах.

Как говорилось в предыдущей лабораторной работе, для повышения привилегий пользователя до уровня администратора системы требуется использовать команду '`su -l`' — с ключом '`-l`' . Данный ключ обеспечивает задание переменных окружения запускаемого от имени суперпользователя интерпретатора команд из настроек суперпользователя — без этого ключа остаются переменные окружения обычного пользователя. Как следствие, командный интерпретатор после этого не сможет найти системные программы, будет записывать временные файлы администратора в каталог

файлов пакетов .i586.rpm), 64-битных процессоров Intel и AMD (с расширением файлов пакетов .x86_64.rpm), архитектуры ARMv5 (с расширением файлов пакетов .arm.rpm) и ARMv7 (с расширением файлов пакетов .armh.rpm), а также архитектуры RISC-V, MIPS, Эльбрус v3 и Эльбрус v4. Кроме того, существуют программы, являющиеся архитектурно-независимыми — например, написанные на интерпретируемых языках. Для того, чтобы избежать дублирования пакетов с такими программами для каждой из архитектур, они упаковываются в пакеты с архитектурой *noarch*. Таким образом, для систем с 32-битными процессорами с системой команд x86 нужно использовать пакеты .i586.rpm и .noarch.rpm, для 64-битных систем — .x86_64.rpm и .noarch.rpm.

Управление пакетами в системе *RPM* осуществляется с помощью команды rpm. Полный формат её вызова можно посмотреть в соответствующем руководстве (man rpm).

Используя команду rpm, можно получать информацию о пакетах, устанавливать, обновлять и удалять их, а также собирать пакеты с исходным кодом и компилировать их в бинарные пакеты. Для получения информации о пакетах предназначается ключ -q.

rpm -q <имя пакета> выведет краткую информацию о версии и релизе установленного пакета:

```
$ rpm -q rpm
rpm-4.0.4-alt77.M40.1
```

Здесь 4.0.4 — версия программы *RPM*, а alt77.M40.1 — релиз пакета.

Более подробную информацию можно получить, добавив ключ -i:

```
$ rpm -qi rpm
Name        : rpm                          Relocations: (not relocateable)
Version     : 4.0.4                             Vendor: ALT Linux Team
Release     : alt77.M40.1                   Build Date: Втр 28 Авг 2007
21:15:45
Install date: Пнд 22 Окт 2007 00:35:45      Build Host:
ldv.hasher.altlinux.org
Group       : System/Configuration/Packaging   Source RPM: rpm-4.0.4-
alt77.M40.1.src.rpm
Size        : 406252                           License: GPL
Packager    : Dmitry V. Levin <ldv@altlinux.org>
URL         : http://www.rpm.org/
Summary     : The RPM package management system
Description :
The RPM Package Manager (RPM) is a powerful command line driven
package management system capable of installing, uninstalling,
verifying, querying, and updating software packages.  Each software
package consists of an archive of files along with information about
the package like its version, a description, etc.
```

Как видно, в пакете указывается, помимо его версии и релиза, также время компиляции пакета, время его установки в системе, лицензия, URL


```
> done
1  :basename
2  :bash
3  :bash2
4  :bunzip2
5  :bzcat
6  :bzip2
7  :bzip2recover
```

Здесь мы получили список файлов из /bin (ls /bin), отфильтровали из него файлы, начинающиеся на b (ls /bin | grep '^b'), и передали полученный список в качестве параметра оператору цикла for. В самом цикле мы вывели текущее значение переменной цикла и номер записи.

Код возврата.

Как говорилось ранее, каждая программа по результату своего выполнения возвращает в операционную систему определённый код возврата. Нулевое значение подразумевает успешное выполнение программы, ненулевое – наличие каких-либо возникших ошибок. Каким ошибка именно соответствует ненулевое значение – определяется самой программой.

Скрипты командного интерпретатора также возвращают коды возврата. По-умолчанию, это код возврата последней выполненной команды скрипта. Есть возможность завершить скрипт с заданным кодом возврата – для этого можно использовать команду exit.

Например, такой скрипт проверит возможность выполнение команды 'su' под текущим пользователем, в случае недостаточности прав – выведет сообщение об ошибке в поток вывода ошибок и завершится с кодом возврата 1:

```
#!/bin/bash
if [ -x /bin/su ]; then
  echo "Нет прав на выполнение команды su" >&2
  exit 1
fi
/bin/su -c 'ls -l'
```

При возможности запуска 'su' будет запрошено выполнение под учётной записью суперпользователя команды 'ls /root'. Код возврата скрипта в этом случае совпадёт с кодом возврата команды 'su', например, если будет неверно введён пароль суперпользователя – код возврата будет содержать ошибку.

При написании скриптов на языке командного интерпретатора, так же как и в программах на других языках программирования, хорошим тоном является проверка успешности выполнения действий, которые могут быть завершены с ошибками, и обработка таких ошибок. Это касается операций с файлами, вызовов внешних программ и т. п.

В данном случае было найдено 5 устаревших пакетов. При выполнении операции `upgrade` система *APT* не устанавливает новые и не удаляет из системы старые пакеты. Поэтому обновить пакет `pam0_passwdqc` она не могла, и предложила обновить только 4 пакета из 5. Подготовив список пакетов, команда `apt-get` задала пользователю вопрос о продолжении операции: '`Do you want to continue? [Y/n]`'. Получив утвердительный ответ (`y`), `apt-get` получила новые версии пакетов и установила их в системе.

Для обновления пакетов, у которых изменились зависимости, служит команда `apt-get dist-upgrade`:

```
# apt-get dist-upgrade
Reading Package Lists... Done
Building Dependency Tree... Done
Calculating Upgrade... Done
The following packages will be upgraded
  pam0_passwdqc
The following NEW packages will be installed:
  libpasswdqc passwdqc-control
1 upgraded, 2 newly installed, 0 removed and 0 not upgraded.
Need to get 52.6kB of archives.
After unpacking 7100B of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 ftp://ftp-distr x86_64/classic passwdqc-control 1.1.0-alt0.4 [6583B]
Get:2 ftp://ftp-distr x86_64/classic libpasswdqc 1.1.0-alt0.4 [30.3kB]
Get:3 ftp://ftp-distr x86_64/classic pam0_passwdqc 1.1.0-alt0.4 [15.7kB]
Fetched 52.6kB in 0s (462kB/s)
Committing changes...
Preparing...              ########################################## [100%]
1: passwdqc-control       ########################################## [100%]
....
3: pam0_passwdqc          ########################################## [100%]
Done.
```

В данном случае у новой версии пакета `pam0_passwdqc` появились зависимости на два новых пакета, которые команда `apt-get` и предложила установить.

Для установки программы используется команда `apt-get install`. В качестве аргумента ей передаются имена пакетов, которые нужно установить. `apt-get` определяет зависимости пакетов и выдаёт полный список всех пакетов, которые будут установлены в системе:

```
# apt-get install rrd-perl
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  libart_lgpl libfreetype libpng12 librrd
The following NEW packages will be installed:
  libart_lgpl libfreetype libpng12 librrd rrd-perl
0 upgraded, 5 newly installed, 0 removed and 0 not upgraded.
Need to get 774kB of archives.
After unpacking 1563kB of additional disk space will be used.
Do you want to continue? [Y/n] n
Abort.
```


сайта разработчика программы, краткое и полное описание программы в пакете.

Список пакетов, установленных в системе, можно получить с помощью команды:

```
$ rpm -qa
vzquota-3.0.9-alt1
mktemp-1.5-alt2
libshhopt-1.1.7-alt4
....
alterator-users-8.0-alt2
kdebase-libkonq-3.5.8-alt11.M40.1
libxine-1.1.10.1-alt1.M40.1
```

Для каждого файла в системе, установленного из пакета, в кэше `rpm` хранится соответствующая запись. Всегда можно посмотреть, какому пакету принадлежит тот или иной файл или каталог. Для этого используется команда `rpm -qf`:

```
$ rpm -qf /usr/bin/rpm
rpm-4.0.4-alt77.M40.1
$ rpm -qf /bin/cp
coreutils-5.97-alt6
$ rpm -qf /home
filesystem-2.3.2-alt1
$ rpm -qf /root
filesystem-2.3.2-alt1
$ rpm -qf /home/student
предупреждение: файл /home/student не принадлежит ни одному из пакетов
```

Как видно, домашний каталог суперпользователя `/root` был создан в системе при установке пакета `filesystem`, а домашний каталог пользователя `student`, разумеется, ни одному из пакетов не принадлежит.

Для установки, обновления и удаления пакетов команду `rpm` использовать не очень удобно. Дело в том, что `rpm` (как и `dpkg`) предназначена для работы с одиночными пакетами. Однако пакеты, как правило, зависят от других пакетов, и для установки пакета требуется также установка и всех тех пакетов, от которых он зависит. Такие зависимости образуют цепочки, и вручную определить весь список необходимых пакетов сложно. Поэтому поверх систем *RPM* и *dpkg* используются системы управления репозиториями пакетов.

Под репозиторием понимается набор пакетов программ, предназначенный для конкретного дистрибутива и связанный общими зависимостями.

Репозитории пакетов существуют практически для всех крупных дистрибутивов. Они подразделяются на официальные, на базе которых и выпускаются дистрибутивы, и неофициальные, поддерживаемые конкретными разработчиками и/или группами разработчиков. Официальные репозитории крупных дистрибутивов насчитывают десятки тысяч пакетов.

- 6 — уровень перезагрузки системы. На этот уровень система переходит по командам `reboot` и `shutdown -r`. После завершения перехода компьютерная система должна перезагрузиться.

Как правило, переходы между уровнями в работающей системе не производятся, и нужны только при её загрузке или остановке.

В ALT Linux и ряде других дистрибутивов сервисы запускаются скриптами, расположенными в каталоге `/etc/rc.d/init.d/`. На этот каталог есть символьная ссылка `/etc/init.d/`, использование путей `/etc/init.d/` и `/etc/rc.d/init.d/` равнозначно. Для управления тем, какой скрипт и на каком уровне запускается, используется команда `chkconfig`.

Посмотреть, какие сервисы должны выполняться при загрузке системы, можно командной `chkconfig --list`:

```
# chkconfig --list
crond           0:off   1:off   2:on    3:on    4:on    5:on    6:off
fbsetfont       0:off   1:off   2:off   3:off   4:off   5:off   6:off
ifrename        0:off   1:off   2:on    3:on    4:on    5:on    6:off
klogd           0:off   1:off   2:on    3:on    4:on    5:on    6:off
lighttpd        0:off   1:off   2:off   3:off   4:off   5:off   6:off
netfs           0:off   1:off   2:off   3:on    4:on    5:on    6:off
network         0:off   1:off   2:on    3:on    4:on    5:on    6:off
portmap         0:off   1:off   2:off   3:off   4:off   5:off   6:off
random          0:off   1:off   2:on    3:on    4:on    5:on    6:off
rawdevices      0:off   1:off   2:off   3:off   4:off   5:off   6:off
sshd            0:off   1:off   2:on    3:on    4:on    5:on    6:off
syslogd         0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

Например, демон `sshd` выключен на уровнях 0, 1 и 6, и включён на уровнях 2, 3, 4 и 5. Посмотреть на состояние конкретного сервиса можно, указав его имя:

```
# chkconfig --list lighttpd
lighttpd        0:off   1:off   2:off   3:off   4:off   5:off   6:off
```

Как видно, сервис `lighttpd` выключен и при перезагрузке системы запускаться не будет.

Для включения сервиса следует выполнить команду `chkconfig <имя сервиса> on`:

```
# chkconfig lighttpd on
# chkconfig --list lighttpd
lighttpd        0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

Выключается сервис командой `chkconfig <имя сервиса> off`.

Включение и выключение сервисов в конфигурации запуска системы не запускает и не останавливает их в работающей системе. Как правило, перезагрузка *nix-систем — это очень редкое и обычно вынужденное событие. Для запуска и остановки сервисов в работающей системе в ALT Linux используется команда `service`. Формат её вызова:

```
service <имя сервиса> <команда>.
```


В данном случае была запрошена установка пакета `rrd-perl`. Этот пакет зависит от библиотеки `librrd`, которая, в свою очередь, использует библиотеки `libpng12` и др. Всего установка пакета потребовала бы установку дополнительно ещё четырёх — что и предложила сделать `apt-get`. Получив отрицательный ответ на вопрос о продолжении операции, `apt-get` отменила её.

Для удаления пакетов используется команда `apt-get remove`. Ей также передаётся список пакетов. Если от удаляемого пакета зависят какие-либо ещё из установленных в системе, `apt-get` предложит удалить их все. При удалении пакетов `apt-get` всегда запрашивает подтверждение операции. Списки удаляемых пакетов следует внимательно просматривать во избежание нежелательных последствий. Например, команда

```
# apt-get remove openssh-server
Reading Package Lists... Done
Building Dependency Tree... Done
The following packages will be REMOVED:
  openssh-server ve-basic
0 upgraded, 0 newly installed, 2 removed and 0 not upgraded.
Need to get 0B of archives.
After unpacking 560kB disk space will be freed.
Do you want to continue? [Y/n] n
Abort.
```

удалит сервер *SSH*. После её выполнения удалённо зайти в систему уже не получится.

В ряде случаев удаляемый пакет критически необходим для системы:

```
# apt-get remove filesystem
Reading Package Lists... Done
Building Dependency Tree... Done
The following packages will be REMOVED:
  SysVinit alternatives apt apt-conf-sisyphus basesystem bash bzip2 bzlib
....
   vitmp vixie-cron zlib
WARNING: The following essential packages will be removed
This should NOT be done unless you know exactly what you are doing!
  apt sed (due to apt) libapt (due to apt) rpm (due to apt)
....
  vitmp (due to basesystem) mktemp (due to basesystem)
0 upgraded, 0 newly installed, 145 removed and 0 not upgraded.
Need to get 0B of archives.
After unpacking 328MB disk space will be freed.
You are about to do something potentially harmful
To continue type in the phrase 'Yes, do as I say!' n
Abort.
```

В данном случае при попытке удалить пакет, содержащий каталоги корневой файловой системы, команда `apt-get` обнаружила 145 зависящих от него пакетов, включая критически необходимые, и выдала соответствующее грозное предупреждение. Отвечать утвердительно на такие вопросы `apt-get` не стоит.

Периодическое (регулярное) выполнение задач.

Скрипты можно использовать для автоматизации тех или иных задач. Очень часто при этом требуется организовать выполнение скрипта в заданное время или через определённые интервалы времени. Для этого существует специальный демон — `crond`.

Для настройки программ на регулярное выполнение используется файл конфигурации, который можно посмотреть командной `crontab -l` и изменить командой `crontab -e`.

Рассмотрим такой файл:

```
$ crontab -l
# DO NOT EDIT THIS FILE - edit the master and reinstall.
# (/tmp/.private/student/crontab.6WaeT9 installed on Mon Mar 17 12:39:10 2008)
# (Cron version V5.0 -- vixie-cron-4.1.20060426-alt3)
#minute (0-59),
#|      hour (0-23),
#|      |       day of the month (1-31),
#|      |       |       month of the year (1-12),
#|      |       |       |       day of the week (0-6 with 0=Sunday).
#|      |       |       |       |       commands
*/1     *       *       *       *       /var/www/bin/log-local.sh
*/2     *       *       *       *       /var/www/bin/log-snmp.sh
```

Строки, начинающиеся с `#` — как обычно, комментарии. Для каждой из запускаемых команд указывается, когда её надо выполнить. Для этого используются пять полей: минуты, часы, дни месяца, месяцы и дни недели. Для каждого из полей можно указать или какое-либо определённое значение, или `*` (астериск), что означает «для всех».

Для выбора дня выполнения задачи можно использовать или поля «день месяца» и «месяц», или поле «день недели». При указании для задачи и дня месяца, и дня недели, эти условия объединяются через логическое сложение (через «логическое ИЛИ»).

Рассмотрим значения этих полей на примера вызова программы `/bin/false`:

`* * * * * /bin/false`	Запускать каждую минуту (каждого часа, каждого дня, каждого месяца, в любой день недели).
`*/3 * * * * /bin/false`	Запускать каждые три минуты (каждого часа, каждого дня, каждого месяца, в любой день недели).
`*/3 1-2 * * * /bin/false`	Запускать каждые три минуты первого и второго часа ночи (каждого дня, каждого месяца, в любой день недели).
`1 1,6 * * * /bin/false`	Запускать в первую минуту первого и

Имя сервиса то же, что и для команды `chkconfig` (и, на самом деле, это имя скрипта из `/etc/init.d/`). Все сервисы поддерживают команды `start` (для запуска неработающего сервиса), `stop` (для остановки работающего сервиса), `restart` (для остановки и последующего запуска сервиса), `status` (для получения статуса сервиса). Возможны и дополнительные команды, которые можно узнать, запустив `service <имя сервиса>` без указания команды.

Подавляющее большинство скриптов в `/etc/init.d` отслеживают состояние запускаемых ими программ и не позволят повторно запустить их.

Например, для сервиса lighttpd:

```
# service lighttpd
Usage: lighttpd {start|stop|restart|condstop|condrestart|condreload|reload|
status}
# service lighttpd status
lighttpd is stopped
# service lighttpd start
Starting lighttpd service: [ DONE ]
# service lighttpd status
lighttpd is running
[root@lab-100 ~]# service lighttpd restart
Stopping lighttpd service: [ DONE ]
Starting lighttpd service: [ DONE ]
# service lighttpd status
lighttpd is running
# service lighttpd stop
Stopping lighttpd service: [ DONE ]
# service lighttpd status
lighttpd is stopped
# service lighttpd restart
Service lighttpd is not running.[PASSED]
Starting lighttpd service: [ DONE ]
#
```

Видно, что сервис `lighttpd` поддерживает команды `start`, `stop`, `restart`, `status`. Команды `condstop`, `condrestart` и `condreload` в основном предназначены для перезапуска сервиса при обновлении пакета с ним.

Изначально сервис не был запущен. По команде `service lighttpd start` он был запущен, что подтвердил последующий вывод команды `service lighttpd status`. Также успешно прошёл перезапуск сервиса (в процессе которого он остановился и заново запустился, перечитав свою конфигурацию), и его остановка. Последняя команда `restart` не нашла работающего сервиса `lighttpd`, о чём сообщила, и потом его успешно запустила.

Помимо *sysvinit* существуют и другие системы инициализации: *Upstart*, *Runit*, *systemd*, и т.д. Выбор той или иной системы инициализации зависит от предпочтений составителей конкретного дистрибутива и направленности конкретного дистрибутива для решения тех или иных задач. Описанная

Внесите произвольные изменения в `~/Documents/file.txt` с помощью редактора `vim`. Для завершения работы с `vim` используйте последовательность команд `<Esc>:wq` .

Изучите список пользователей и групп, находящийся в файлах `/etc/passwd` и `/etc/group`. Изучите права на файлы в домашнем каталоге пользователя, каталогах */etc*, */sbin*, */var/log*. Попробуйте прочитать записи в системном журнале `/var/log/messages` .

Получите права суперпользователя, используя команду `su -l`. Ключ `-l` обязателен. Рекомендуется выйти из Midnight Commander перед запуском `su`. Изначально пароль пользователя `root` отсутствует, после задания пароля `su` будет его запрашивать. Без указания выполняемой команды в качестве параметра `su` запускает командный интерпретатор с правами суперпользователя. Приглашение для `root` выглядит так:

```
[root@lab-100 ~]#
```

Запустите командный интерпретатор с правами `root`.

Задайте пароль на пользователя `root`. Задайте пароль для пользователя `student`, запустив `passwd` с соответствующим параметром.

Завершите сеанс `root`, выйдя из командного интерпретатора (`exit` или `<Ctrl>+<D>`).

Снова запустите командный интерпретатор с правами `root`.

Создайте нового пользователя. Имя пользователя выберите самостоятельно. Имя пользователя может содержать латинские строчные буквы, цифры и символы - (дефис) и _ (нижнее подчёркивание), и по возможности не должно превышать 8-ми символов. Для создания пользователя используйте команду `useradd`.

Проверьте список пользователей и групп в системе.

Проверьте, какие пользователи имеют право на запуск команды `su` (полное имя файла команды — `/bin/su`). Внесите созданного пользователя в нужные группы, отредактировав файл `/etc/group`.

Задайте пароль для созданного пользователя.

Запустите вторую терминальную сессию. Для этого в меню окна *PuTTY* (доступного при нажатии на иконку приложения слева в заголовке окна) выберите пункт *New Session*. Повторите настройки подключения и осуществите вход в систему под учётной записью созданного пользователя. Убедитесь в возможности получения им прав суперпользователя, запустив команду `su -l`.

	шестого часа ночи, т.е. в 01:01 и 06:01 (каждого дня, каждого месяца, в любой день недели).
`1 1 * * 1 /bin/false`	Запускать в 01:01 каждый понедельник.
`1 1 * 2 1 /bin/false`	Запускать в 01:01 каждый понедельник или в 01:01 каждого дня февраля.
`* * 31 10 5 /bin/false`	Запускать каждую минуту каждого часа 31 октября, или в каждую минуту каждого часа каждой пятницы.

При выполнении по `cron`'у задач, которые потенциально могут выполняться длительное время, следует предусмотреть и блокировать повторный запуск `cron`'ом скрипта в то время, когда ещё не успел завершиться предыдущий. Обычно такое можно делать, создавая и анализируя при запуске скрипта файл блокировки. Например:

```
$ cat lock.sh
#!/bin/bash
LOCK=/tmp/file.lock

if [ -f "$LOCK" ]; then
        echo 'Скрипт уже работает'
        exit 1
fi
touch "$LOCK"

sleep 1m

rm -f "$LOCK"
```

Здесь при запуске скрипта проверяется существование файла, и если он существует, то выполнение скрипта завершается. Иначе файл создаётся, выполняется некое действие (в данном случае — просто ожидание на 1 минуту), и перед завершением работы файл блокировки удаляется.

Пример выполнения:

```
$ ./lock.sh &
[1] 7702
$ ./lock.sh &
[2] 7704
$ Скрипт уже работает
[2]+  Exit 1                  ./lock.sh
```

Для получения данных предлагается использовать следующие программы:

log-local.sh — получение и запись в файл локальной статистики.

```
#!/bin/bash
# Script for logging current system status:
# - number of processes
# - RX and TX bytes over venet0 network interface

# Log to this file:
LOG_FILE=/var/www/stat/local.log

# Timestamp:
TS=`date '+%Y-%m-%d %H:%M:%S'`

# Process number
PROCNUM=`ps aux | wc -l`
PROCNUM=$(($PROCNUM-1))

# netstat info
NETBYTES=`netstat -i | grep '^eth0' | awk '{print "RX ",$4,"bytes, TX ",
$8,"bytes."}'`

# Log all to the file
echo "$TS => Procs: $PROCNUM, $NETBYTES" >> "$LOG_FILE"
#------------------------------------------------------------
```


Удалите учётную запись пользователя `student`, используя команду `userdel`. Убедитесь в успешном выполнении команды, проверив содержимое файлов `/etc/passwd` и `/etc/group`, а также попробовав запустить терминальную сессию под этим пользователем.

Найдите в `/home` домашние каталоги созданного и удалённого пользователей. Перенесите созданный в `Documents/` текстовый файл в каталог созданного пользователя.

При необходимости поменяйте права на файл с помощью команд `chmod` и `chown`.

Удалите домашний каталог пользователя `student`.

Получите полный список пакетов *RPM*, установленных в системе, командой `rpm -qa`. Получите детальную информацию об одном или нескольких пакетах, выполнив команды `rpm -qi <имя пакета>`.

Рассмотрите настройки списка репозиториев системы *APT*, находящиеся в файле `/etc/apt/sources.list`. Внесите в него записи о репозиториях, согласно выданным преподавателем рекомендациям.

Обновите локальные списки пакетов системы *APT*, выполнив команду `apt-get update`. В случае появления сообщений об ошибках проверьте и исправьте список репозиториев, и снова выполните обновление списка пакетов.

Обновите систему до текущего состояния репозиториев, выполнив команды `apt-get upgrade` и `apt-get dist-upgrade`. Обратите внимание на перечень обновлённых пакетов. Получите информацию о последних изменениях какого-либо пакета, выполнив команду `rpm -q --changelog <имя пакета>`.

Используя команду `apt-cache search <строка для поиска>`, найдите пакет, содержащий веб-сервер `lighttpd`. Установите пакет через вызов команды `apt-get install`.

Найдите в основном конфигурационном файле веб-сервера `lighttpd` (`/etc/lighttpd/lighttpd.conf`) путь к каталогу с файлами, доступными веб-серверу (параметр `server.document-root`).

Поместите в указанный каталог (при необходимости создав его) произвольный текстовый файл. Имя файла должно иметь расширение `.txt`.

Браузер подключается к веб-серверу, устанавливая сетевое соединение по протоколу TCP/IP. Выбор нужного веб-сервера осуществляется по определённому адресу IP и порту TCP. По-умолчанию, для схемы http://

Для удобного доступа к различным скриптам в `/var/www/html` предлагается разместить индексный файл с названием `index.html` вида:

```
<html>
<head>
<title>index</title>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
</head>
<body>
<h1>Текущая статистика</h1>
<ul>
<li><a href="/cgi-local.sh">Простой скрипт статистики локальной
системы</a></li>
<li><a href="/cgi-local-html.sh">HTML-скрипт статистики локальной
системы</a></li>
<li><a href="/cgi-local-html-table.sh">HTML-скрипт с выводом таблицей
статистики локальной системы</a></li>
<li><a href="/cgi-local.rrd">Выдача графиков статистики локальной
системы</a></li>
<li><a href="/cgi-snmp.sh">Простой скрипт статистики интерфейса SNMP</a></li>
<li><a href="/cgi-snmp-html.sh">HTML-скрипт статистики интерфейса
SNMP</a></li>
<li><a href="/cgi-snmp-html-table.sh">HTML-скрипт с выводом таблицей
статистики интерфейса SNMP</a></li>
<li><a href="/cgi-snmp.rrd">Выдача графиков статистики интерфейса
SNMP</a></li>
</ul>
</body>
</html>
```

Исходные тексты скриптов для сбора данных, скриптов для форматирования и вывода собранных данных, и приведённого выше индексного файла доступны для просмотра и скачивания на сайте http://edu.cbias.ru. Там же, на странице http://lab-00.edu.cbias.ru/ , можно посмотреть примеры результатов работы этих скриптов.

log-snmp.sh — получение и запись в файл SNMP-статистики.

```
#!/bin/bash
# Script for logging current SNMP information:
# - RX and TX bytes over some network interface

# Log to this file:
LOG_FILE=/var/www/stat/snmp.log

# Network interface number:
N=8

# SNMP host
HOST=192.168.222.100

# SNMP community
COMMUNITY=public

# MIBS
MIB1="IF-MIB::ifDescr.$N"
MIB2="IF-MIB::ifInOctets.$N"
MIB3="IF-MIB::ifInUcastPkts.$N"
MIB4="IF-MIB::ifOutOctets.$N"
MIB5="IF-MIB::ifOutUcastPkts.$N"

############################
# Timestamp:
TS=`date '+%Y-%m-%d %H:%M:%S'`

# snmp info
RES=''

for MIB in $MIB1 $MIB2 $MIB3 $MIB4 $MIB5; do
    LINE=`snmpget -c $COMMUNITY -v 1 $HOST $MIB`

    NAME=`echo $LINE | sed "s/^IF-MIB::\([[:alnum:]]\+\).*/\1/"`
    VALUE=`echo "$LINE" | sed "s/^IF-MIB::[[:alnum:]]\+\.$N = [[:alnum:]]\
+: //"`

    RES="$RES  $NAME:$VALUE"
done

# Log all to the file
echo "$TS => $RES" >> "$LOG_FILE"
#-------------------------------------------------------------
```

Запуск скриптов получения данных предполагается осуществлять раз в минуту для получения локальной информации, и раз в две минуты — для получения информации с коммутатора через *SNMP*.

Задания на лабораторную работу.

1. Выполнить удалённую регистрацию в системе.
2. Провести ознакомление с операционной системой. Изучить структуру каталогов сервера. Посмотреть доступные команды в системе, вызвать справочное руководство по каким-либо из них. Создать текстовый файл, используя редактор `vi`.
3. Используя команду `su`, получить привилегии суперпользователя системы.
4. Изменить пароли пользователя и суперпользователя системы.
5. Создать новую учётную запись пользователя.
6. Зарегистрироваться в системе под созданным в п. 5 пользователем, убедиться в возможности использования им команды `su`.
7. Удалить учётную запись пользователя `student`.
8. Изучить список пакетов, установленных в системе.
9. Настроить список репозиториев пакетов для системы *APT*. Провести обновление системы до текущего состояния репозитория.
10. Установить веб-сервер `lighttpd`, запустить сервер. Проверить работу веб-сервера. Настроить его автоматический запуск при загрузке системы.
11. Перезагрузить систему. Убедиться, что веб-сервер `lighttpd` автоматически запустился после перезагрузки системы.
12. Доставить в систему всё необходимое для работы скриптов сбора и отображения статистики программное обеспечение.
13. Адаптировать приведённые в описании работы скрипты, получающие значения статистических параметров и записывающие их журналы.
14. Обеспечить периодическое регулярное выполнение скриптов.
15. Адаптировать приведённые в описании работы скрипты для отображения записываемых в пп. 13-14 данных из журналов, обеспечить их выполнение из командной строки.
16. Настроить `lighttpd` для удалённого обращения из браузера к указанным скриптам и отображения собираемых данных в веб-браузере на удалённом рабочем месте.
17. Обеспечить безопасное выполнение скриптов.