Теоретические сведения.

Основные понятия операционных систем семейства *nix.

Первая система UNIX была разработана в 1969 - 1970 годах в подразделении Bell Labs компании AT&T. В 1973 году система была почти полностью переписана на также разработанном в Bell Labs языке высокого уровня C. Это позволило легко переносить UNIX на вычислительные системы различных архитектур и способствовало широкому её распространению. На настоящий момент в мире создано и используется большое число различных UNIX-систем. С юридической точки зрения только часть из них имеет право называться «UNIX», остальные же, хотя и используют сходные концепции и технологии, объединяются термином «UNIX-подобные» (*англ.* Unix-like). Для краткости всё семейство операционных систем класса Unix принято обозначать как *nix.

Некоторые отличительные признаки UNIX-систем включают в себя:

- использование простых текстовых файлов для настройки и управления системой;
- широкое применение утилит, запускаемых в командной строке;
- взаимодействие с пользователем посредством виртуального устройства — терминала;
- представление физических и виртуальных устройств и некоторых средств межпроцессового взаимодействия как файлов;
- использование конвейеров из нескольких программ, каждая из которых выполняет одну задачу.

Технически операционные системы *nix состоят из ядра системы и различных утилит и программ. Ядро обеспечивает общий интерфейс к оборудованию, управление выполнением программ, разделение между ними аппаратных ресурсов компьютерной системы. Основной концепцией, заложенной в архитектуру *nix, является представление различных аппаратных устройств как файлов и предоставление программам возможности работать с устройствами как с файлами.

Одной из ключевых особенностей операционных систем *nix является наличие большого количества разнообразных программ-утилит. Такие программы, запускаемые в командной строке, предназначены для выполнения определённого элементарного действия в системе — например, вывода текстового файла на экран, вывода содержимого каталога, записи текста в файл. Операционные системы *nix предоставляют удобные и гибкие механизмы объединения работы таких отдельных простых программ для выполнения конкретных задач пользователей. В данной лабораторной работе проводится рассмотрение и изучение этих механизмов.

тур данных внутри ядра *nix, как к файлам. В частности, в *procfs* можно посмотреть текущий список выполняющихся процессов, состояние оборудования, настройки и текущее состояние сетевых устройств, и т.п. Программы, предназначенные для вывода подобной информации, получают её из *procfs*. Кроме того, ряд файлов в *procfs* доступны для записи, и с их помощью можно изменить параметры работы ядра *nix.

- *sysfs* — файловая система, работающая со структурами ядра Linux и позволяющая получить данные об оборудовании системы. В частности, с использованием *sysfs* производится конфигурация устройств «горячего» подключения.
- *udevfs* — файловая система, предназначенная для хранения файлов устройств, с поддержкой создания/удаления файлов устройств «горячего» подключения.
- *tmpfs* — файловая система, предназначенная для хранения файлов в виртуальной памяти. Основное назначение системы — размещение временных файлов, которые можно потерять при перезагрузке системы. Использование *tmpfs* на современных компьютерных системах, с большим объемом ОЗУ и достаточно большим размером файлов подкачки, позволяет существенно ускорить, например, выполнение компиляции и сборки сложных программных продуктов. Кроме того, *tmpfs* используется во встраиваемых системах.
- *jffs2* (*Journalling Flash File System version 2*) — файловая система, оптимизированная для работы с программируемой Flash-памятью (ППЗУ) с учётом особенностей износа Flash-памяти при регулярной записи в неё данных. Применяется во встраиваемых системах для хранения настроек.
- *squashfs* — файловая система, обеспечивающая хранение данных и структур каталогов в сжатом состоянии. Предназначена только для чтения данных, широко используется во встраиваемых системах.

Существуют также сетевые файловые системы. Эти файловые системы предназначены для доступа к информации, хранящейся на других системах, через компьютерную сеть. Из сетевых файловых систем можно отметить:

- *CIFS* (*Common Internet File System, старое название SMB, Server Message Block*) — сетевая файловая система, использующаяся в сетях Microsoft Windows.
- *NFS* (*Network File System*) — сетевая файловая система, изначально появившаяся для систем *nix. По сравнению с *CIFS* существенно более простая в реализации, но и с существенно менее гибким управлением доступом к файлам.

Современные файловые системы организуют хранение файлов в иерархической структуре каталогов. Все перечисленные выше файловые

системы поддерживают длинные имена файлов. Как правило, максимальная длина имени файла составляет 255 символов — т.е., при использовании кодировки UTF-8, 127 символов русского алфавита. В именах файлов и каталогов допускаются любые символы, кроме символов `NULL` (символ с кодом ASCII 0) и `/` (слеш). Символ `NULL` используется как ограничитель строки, он следует за последним значащим символом строки переменной длины. Символ `/` (слеш) служит разделителем имён каталогов при указании пути к файлу. Отметим, что в операционных системах семейства Microsoft Windows для этой цели используется символ `\` (обратный слеш).

В имени файла может содержаться расширение — несколько (обычно до четырёх) символов, отделённых от основной части имени файла символом `.` (точка). Обычно через расширение указывается формат файла. К широко используемым расширениям относятся:

- `.jpg`, `.gif`, `.png`, `.tiff` — для графических файлов;
- `.html`, `.htm` — для файлов в формате HTML;
- `.pdf`, `.ps` — для файлов PDF и PostScript;
- `.c`, `.h`, `.cpp` — для исходных текстов и заголовков программ на C и C++;
- `.o` — для скомпилированного объектного кода;
- `.sh` — для скриптов командного интерпретатора;
- `.tar` – для файловых архивов в формате утилиты tar;
- `.gz`, `.bz2`, `.zip`, `.xz` – для файлов, сжатых утилитами `gzip`, `bzip2`, `zip` и `xz` соответственно, и т.п.

Часто в расширении указывается дополнительная информация — например, для библиотек в виде расширения может указываться номер версии библиотеки. Однако для самой операционной системы расширения файлов никакой смысловой нагрузки не несут, никаких ограничений ни на размер расширений, ни на их число не накладывается.

В *nix строчные и заглавные буквы в именах файлов различаются, т.е. файлы `file.txt`, `file.TXT`, `File.txt` — три разных файла, которые могут сосуществовать в одном каталоге. Однако могут быть и исключения — например, для файловой системы *VFAT* — это один и тот же файл, а для *CIFS* поведение зависит от настроек сервера *CIFS*.

В каждой файловой системе имеется каталог верхнего уровня. Существует понятие корневой файловой системы — это каталог верхнего уровня файловой системы системного диска. На системном диске размещаются основные файлы операционной системы и с него выполняется загрузка системы. Корневой каталог имеет путь `/`. Для каждого файла в системе можно указать полный путь — перечисление иерархии всех каталогов от корневого каталога до самого файла.

ются при монтировании файловой системы, для других (*XFS*) — при необходимости при проверке файловой системы.

- `/media/` — содержит каталоги, в которые монтируются при подключении съёмные носители данных. Для использования какого-либо диска (включая оптические диски CD/DVD, дискеты, накопители USB Flash) его файловая система должна быть смонтирована в общее дерево. Это может делаться как вручную (в подкаталоги `/mnt/`), так и автоматически, при использовании соответствующих системных утилит. Эти утилиты, при подключении нового диска к системе, создают для него подкаталог в каталоге `/media/` и монтируют в него файловую систему диска. Стоит обратить внимание на то, что вынуть съёмный диск из компьютера можно только после его размонтирования (отключения от общей файловой системы). Для ряда съёмных накопителей (например, дисков CD/DVD) система не позволит их извлечь без размонтирования их файловой системы. Размонтировать файловую систему можно только тогда, когда ни одна из работающих программ не обращается к съёмному диску. Если диск размонтировать не получается, надо найти и завершить такие программы. Если просто вынуть дискету или отключить USB Flash, то можно нарушить работу операционной системы и повредить файловую систему дискеты / накопителя USB Flash.
- `/mnt/` — содержит каталоги, в которые временно и вручную монтируются различные диски.
- `/opt/` — каталог для крупных и независимых программных пакетов. К таким пакетам обычно относятся коммерческие продукты, которые не могут использовать общесистемные настройки и библиотеки. Они размещаются в отдельных подкаталогах внутри каталога `/opt/`, со своими подкаталогами `etc/`, `bin/`, `lib/`, `var/`, `sbin/` и т.п;
- `/proc/` — каталог, в который монтируется файловая система *procfs*. Содержит информацию о состоянии системы;
- `/root/` — домашний каталог суперпользователя. Поскольку каталоги пользователей системы могут быть размещены на отдельных дисках, домашний каталог суперпользователя отделён от них и размещается в корне файловой системы, на системном диске;
- `/sbin/` — содержит основные системные утилиты. В отличие от утилит из `/bin/`, эти утилиты не предназначены для использования обычными пользователями, но также жизненно необходимы для загрузки системы;
- `/srv/` — каталог, предназначенный для сервисов системы. Обычно никак не используется;
- `/sys/` — каталог, в который монтируется файловая система *sysfs*. Содержит информацию об оборудовании системы;
- `/tmp/` — содержит временные файлы. Может быть смонтирован как *tmpfs*, с удалением всего содержимого при перезагрузке машины. В каталоге `/tmp/` может создавать файлы и подкаталоги любой пользователь

системы;

- `/usr/` — предназначен для размещения прикладных программ и их статических данных. Подкаталоги внутри `/usr/` образуют вторичную структуру каталогов, повторяющую структуру корневого каталога, и содержат:
  - `/usr/bin/` — каталог для прикладных программ, доступных пользователям;
  - `/usr/include/` — каталог для файлов заголовков C;
  - `/usr/lib/` — каталог для библиотек, использующихся программами в `/usr/bin/` и `/usr/sbin/`;
  - `/usr/sbin/` — каталог с некритичными системными программами — например, с различными сетевыми серверами;
  - `/usr/share/` — каталог с архитектурно-независимыми данными, в т.ч.:
    - `/usr/share/doc/` — каталог с документаций для пакетов программ;
    - `/usr/share/man/` — каталог со страницами справочных руководств по программам;
    - `/usr/share/info/` — каталог со справочными руководствами в формате info, предназначенных для просмотра одноимённой командой;
  - `/usr/src/` — каталог с исходными текстами программ;
  - `/usr/X11R6/` — структура каталогов третьего уровня для программ *X Window System, Version 11, Release 6* — в т.ч. `/usr/X11R6/bin/`, `/usr/X11R6/lib/`, `/usr/X11R6/man/`, и т.д.;
  - `/usr/local/` — структура каталогов третьего уровня для программ, устанавливаемых из исходных текстов. Содержит подкаталоги `/usr/local/bin/`, `/usr/local/lib/`, `/usr/local/sbin/` и т.д.;
- `/var/` — предназначен для различных изменяемых при работе системы файлов. Внутри `/var/` находятся:
  - `/var/cache/` — файлы с кэшированными данными приложений, например, полученные из сети пакеты программ, отформатированные и готовые для показа пользователю страницы справочных руководств, и т.п.;
  - `/var/lock/` — файлы блокировки, предназначенные для отслеживания использования ресурсов в системе;
  - `/var/log/` — журналы (*логи*). В файлы в этом каталоге выводится информация от работающих в системе неинтерактивных программ;
  - `/var/spool/` — каталог с данными, которые ожидают обработки,

`chuser()` свой идентификатор пользователя на непривилегированный.

В качестве дополнительной меры безопасности существует возможность изменения в настройках запущенных процессов указателя на корневой каталог. В этом случае работающей программе доступно не всё дерево каталогов системы, а только некоторая его часть, обратится к файлам за пределы которой программа не может. Обычно при этом для программы создаётся каталог с именем `/var/lib/<имя программы>`, с подкаталогами `etc/`, `lib/`, `tmp/`, `var/`, в которые копируется минимально необходимый для работы данной программы набор конфигурационных файлов и системных библиотек. Смена общего корневого каталога на каталог `/var/lib/<имя программы>` выполняется системным вызовом `chroot()`.

Системные вызовы `chuser()` и `chroot()` доступны только суперпользователю, поэтому после перехода в непривилегированный режим работы процесс не может вернуть себе полномочия `root` обратно. Все создаваемые им процессы также будут работать с правами псевдопользователя. В случае, если в программе будет обнаружена уязвимость, и злоумышленник получит над ней контроль (т.е. сможет запускать свои программы в системе), он не сможет получить права суперпользователя и выйти за границы *chroot-окружения*.

Как правило, идентификатор (*UID*) псевдопользователя меньше 500, а обычного пользователя — равен или больше. Однако данное разделение чисто условное и соблюдается по договорённости.

Для интерактивной работы пользователей с системой используется понятие терминала — устройства, с которого поступают вводимые пользователем команды, и на который выводится результат их выполнения. Для начала работы с системой пользователь должен зарегистрироваться на одном из поддерживаемых системой терминальных устройств. При локальной работе терминалом являются подсоединённые к системе монитор и клавиатура. В случае удалённых сеансов работы, пользователь должен на своём рабочем месте запустить программу — эмулятор терминала и соединиться с удалённой системой. Разумеется, на удалённой системе при этом должен быть запущен соответствующий сервер, обеспечивающий такие соединения.

В начале работы с системой, система запрашивает имя пользователя и его пароль. При совпадении введённых значений со значениями, хранящимися в учётной записи в `/etc/passwd`, система разрешает работу пользователя с данным терминалом и запускает на нём командный интерпретатор, позволяя вводить команды.

При этом нет ограничений на число параллельно работающих с системой пользователей — каждый работает независимо в своём терминале. Можно одновременно открыть и несколько терминальных сессий с системой для одного и того же пользователя, и работать одновременно с несколькими терминалами.

Права доступа к файлам.

Поскольку система Linux с самого начала разрабатывалась как многопользовательская, в ней предусмотрен такой механизм, как права доступа к файлам и каталогам. Он позволяет разграничить полномочия пользователей, работающих в системе. В частности, права доступа позволяют отдельным пользователям иметь «личные» файлы и каталоги. Например, если пользователь `student` создал в своём домашнем каталоге файлы, то он является владельцем этих файлов и может определить права доступа к ним для себя и остальных пользователей. Он может, например, полностью закрыть доступ к своим файлам для остальных пользователей, или разрешить им читать свои файлы, запретив изменять и исполнять их.

Правильная настройка прав доступа позволяет повысить надёжность системы, защитив от изменения или удаления важные системные файлы. Наконец, поскольку внешние устройства с точки зрения системы также являются объектами файловой системы, механизм прав доступа применяется и для управления доступом к устройствам.

С точки зрения самой системы работа пользователя в ней — это выполнение программ (процессов) с идентификаторами UID/GID пользователя, которые осуществляют различные действия с файлами и каталогами, и запускают на выполнение другие процессы. Например, одна из таких программ — командная оболочка, которая считывает команды пользователя из командной строки и передаёт их системе на выполнение.

Каждая программа (процесс) выполняется от имени определённого пользователя (т. е. с определёнными идентификаторами UID/GID). Её возможности работы с файлами и каталогами определяются правами доступа, заданными для этого пользователя.

Содержимое файла программа может считывать или записывать, а если в файле хранится другая программа, то её можно запустить на выполнение и создать новый процесс. Из каталога можно считать список содержащихся в нём файлов и каталогов, или внести в этот список изменения — создать новую запись (файл или каталог), переименовать или удалить существующую. Кроме того, в каталог можно перейти — сделать его текущим для данного процесса.

У любого файла в системе есть владелец — один из пользователей. Однако каждый файл одновременно принадлежит и некоторой группе пользователей системы.

Права доступа определяются по отношению к трём типам действий: чтение, запись и исполнение. Эти права доступа могут быть предоставлены трём классам пользователей: владельцу файла (пользователю), группе, которой принадлежит файл, а также всем остальным пользователям, не входящим в эту группу. Право на чтение даёт пользователю возможность читать содержимое файла или, если такой доступ разрешён к каталогам, просматривать содержимое каталога (используя команду `ls`). Право на запись даёт пользователю возможность записывать или изменять файл, а право на

Для /var/nobody первое поле содержит drwxr-x---. Это каталог, владелец которого — root — имеет права rwx. Группа — nobody — имеет права r-x, т.е. может переходить в каталог и читать его. Прочие пользователи доступа к каталогу не имеют. Такие права объясняются тем, что nobody — это псевдопользователь, и /var/nobody — его домашний каталог (см. запись в /etc/passwd). Это бесправный пользователь, и даже прав на запись чего-либо в свой домашний каталог у него нет.

Для /var/mail в первом поле стоит lrwxrwxrwx. Первый символ l означает символьную ссылку. Согласно выводу команды ls -l, эта ссылка указывает на /var/spool/mail (путь spool/mail указан относительно каталога, где размещена ссылка — т.е. /var). Права доступа к файлу или каталогу, на который ссылается символьная ссылка, определяются правами на сам файл, а не правами ссылки. Поэтому здесь права доступа ничего не означают.

Также видно два особых случая. Первый — это /var/tmp. Права на этот каталог — rwxrwxrwt. Последний символ t означает наличие у каталога дополнительного флага — т.н. *sticky bit*. Это каталог для временных файлов, и в него разрешена запись всем пользователям. Однако удалять из него пользователи могут только свои файлы.

Второй случай — это /bin/su . Здесь права — rws--x---. Владелец файла (root) может его читать, записывать и запускать. Пользователи, включённые в группу wheel, могут только запускать этот файл, прочитать его и, тем более, записать в него они не имеют права. Все прочие пользователи никаких прав на этот файл не имеют. Буква s вместо x для прав владельца файла имеет особый смысл. Это т.н. *SUID bit*, и его наличие означает, что данная программа будет запускаться не с правами пользователя, а с правами владельца файла. Иными словами, непривилегированный пользователь (но входящий в группу wheel!) может запустить эту программу и получить права её владельца — т.е. суперпользователя.

Как правило, настройки современных Linux-систем в целях повышения безопасности запрещают удалённый вход в систему с правами суперпользователя, в ряде дистрибутивов для пользователя root запрещён и локальный вход в систему. Программа /bin/su является одним из способов повысить права обычного пользователя до администратора системы. Именно поэтому её выполнение разрешено только пользователям из группы wheel.

Возможность доступа к файлу зависит также от прав доступа к каталогу, в котором находится файл. Например, даже если права доступа к файлу установлены как rwxrwxrwx, другие пользователи не могут получить доступ к файлу, пока они не имеют прав на исполнение для каталога, в котором находится файл. Другими словами, чтобы воспользоваться имеющимися у вас правами доступа к файлу, вы должны иметь право на исполнение для всех каталогов вдоль пути к файлу. Например, псевдопользователь nobody не сможет прочитать файл ~/file.txt, несмотря на то, что права на этот

файл — rw-r--r--, т.к. права доступа к домашнему каталогу /home/student/ — rwx------.

Установка и поддержание оптимальных прав доступа является одной из важнейших задач системного администратора. Права должны быть достаточными для нормальной работы пользователей и программ, но не большими, чем необходимо для такой работы. Дистрибутивы ALT Linux обладают продуманной системой прав (предопределённые группы, псевдопользователи для различных программ-серверов, права доступа для системных файлов и каталогов). Прежде чем вносить существенные изменения в эту систему, целесообразно понять её логику и выяснить, нет ли другого способа достичь нужной цели.

Поскольку программы, исполняемые от имени суперпользователя (root), могут совершать любые действия с любыми файлами и каталогами, их выполнение может нанести системе серьёзный ущерб. Это может быть как следствием уязвимостей или ошибок в программах, так и результатом ошибочных действий самого пользователя. Поэтому работа с правами суперпользователя требует особой осторожности.

Понятие командного интерпретатора.

Чтобы обеспечить взаимодействие пользователя с операционной системой и с прикладными программами необходим интерфейс: система передачи команд пользователя операционной системе и ответов системы обратно пользователю. Такое взаимодействие представляет собой «диалог» пользователя с компьютером на специальном языке, будь то язык, использующий знаки, похожие на слова и высказывания естественного языка, или язык изображений. На сегодня известны две принципиальные возможности организации интерфейса: графический интерфейс и командная строка.

Командная строка — приглашение оболочки, обозначающее готовность системы принимать команду пользователя — в наиболее явной форме демонстрирует идею диалога. На каждую введенную команду пользователь получает ответ от системы: либо очередное приглашение, означающее, что команда выполнена, и можно вводить следующую, либо сообщение об ошибке, представляющее собой высказывание системы о произошедших в ней событиях, адресованное пользователю. При работе в операционной среде с графическим интерфейсом происходящий диалог пользователя с системой не столь очевиден, хотя с точки зрения системы клик мышью в определенной области на экране аналогичен команде, введенной с клавиатуры, а ответ системы пользователю может быть представлен в виде диалогового окна.

При работе с командной строкой для организации интерфейса используются специальные программы — командные интерпретаторы. Они принимают от пользователя выдаваемые им команды в виде строк текста, содержащих имена программы и параметры, с которыми эти программы

Основные команды системы.

Перед рассмотрением команд системы стоит отметить возможности редактирования командной строки в bash. Указанием на то, что командный интерпретатор готов принимать команды, служит т.н. приглашение — строка вида:

```
[student@lab-100 ~]$
```

В ней указывается имя пользователя и системы, на которой выполняется интерпретатор, текущий каталог (в данном случае — ~ (тильда), что означает сокращение для домашнего каталога пользователя). Завершает приглашение символ $ (знак доллара). Для суперпользователя таким символом является # (октоторп).

Команда набирается как обычная строка в текстовом редакторе. Перемещать курсор по строке возможно с помощью клавиш управления им, работают клавиши <Home> (начало строки), <End> (конец строки), <Del> (для удаления символа под курсором) и <Backspace> (для удаления символа перед курсором). Клавиша <Tab> имеет особый смысл — при её нажатии bash попытается дополнить текущее слово до ближайшего имени файла. Т.е., если в каталоге есть файлы a.txt, a1.txt и b.txt, ввод b и нажатие <Tab> дополнит b до b.txt. Если bash не может однозначно дополнить строку (например, при вводе a и <Tab>), то повторное нажатие на <Tab> выдаст все возможные варианты (в данном случае — a.txt и a1.txt). Клавишей <Tab> можно дополнять и команды, поскольку они для системы также являются файлами.

bash поддерживает историю вводимых команд. Перемещаться по списку команд можно клавишами управления курсором <стрелка вверх> и <стрелка вниз>. Команды из истории можно редактировать, как и обычные. Т.е., если bash не распознал введённую команду и выдал ошибку, проще не набирать команду заново, а, нажав <стрелку вверх>, вызвать последнюю команду и отредактировать её.

Получить полный список команд из истории можно командой history. В качестве необязательного параметра можно задать число последних команд, которые и будут выведены на экран. В истории команд по-умолчанию хранится до 500 последних команд, в системе они сохраняются в файле ~/.bash_history .

При работе с системой один из каталогов является текущим. В начале сеанса работы текущим каталогом становится домашний каталог пользователя. В приведённых ниже примерах $ или # означают приглашение командной строки. Вводить его не нужно. <имя файла> — имя произвольного файла в системе, абсолютное или относительное. Абсолютные имена файлов начинаются с символа / (слеш), включают в себя все родительские каталоги и отсчитываются от корня файловой системы. Относительные имена не начинаются с / и отсчитываются от

текущего каталога.

Текущий каталог может быть изменён командой cd:

$ cd	Перейти в домашний каталог.
$ cd /home	Перейти в каталог /home/.
$ cd ..	Перейти в каталог одним уровнем выше текущего.
$ cd ~	Перейти в домашний каталог (~ (тильда) — сокращение для обозначения домашнего каталога).
$ cd ~/Documents	Перейти в подкаталог Documents/ домашнего каталога.
$ cd ../../etc	Перейти в каталог ../../etc с использованием относительного пути. При текущем каталоге /home/student/ эта команда позволит перейти к каталогу /etc/.

Посмотреть текущий каталог можно командой pwd.

Для просмотра каталогов используется команда ls:

$ ls	Получить список файлов в текущем каталоге.
$ ls -l	Получить список файлов в полном формате, с указанием их прав, владельцев, групп, размера, даты создания и пр.
$ ls -l /etc	Получить список файлов в полном формате в каталоге /etc/.
$ ls -a	Получить список всех файлов в текущем каталоге. По-умолчанию, без флага -a, ls не показывает файлы, начинающиеся с символа . (точка). В таких файлах обычно хранятся настройки программ пользователя.
$ ls -al	Получить список всех файлов в полном формате в текущем каталоге.

Для создания каталога используется команда mkdir <имя каталога>.

Для удаления пустого каталога используется команда rmdir <имя каталога>.

Для удаления файла используется команда rm. Отменить результат выполнения команды rm и восстановить удалённые из системы файлы практически нельзя, штатных средств для этого не предусмотрено.

$ rm file.txt	Удалить файл file.txt в текущем каталоге.
$ rm *.txt	Удалить все файлы, заканчивающиеся на .txt, в текущем каталоге.

ческом режиме. Если пароль покажется программе passwd слишком слабым, она не позволит его задать.

Учитывая особый статус учётной записи пользователя root, в Linux-системах обычно удалённый вход этого пользователя запрещён.

Как указывалось выше, для получения справки по любой команде используется команда man. Для получения справки по использованию man следует ввести man man.

Для удобного перемещения по дереву каталогов и работы с файлами возможно использование файловых менеджеров. Наиболее распространённый из них — *Midnight Commander*, запускаемый командой mc. Он использует стандартный двухпанельный интерфейс файловых менеджеров типа *Norton Commander*, встроенный текстовый редактор, программу просмотра текста, может работать с архивами в различных форматах, с файлами на удалённых серверах *ftp*, *cifs*, *ssh* и др.

В *Midnight Commander* существует встроенный интерфейс командной строки, вызываемый комбинацией клавиш <Ctrl>+<o>. Повторное нажатие <Ctrl>+<o> возвращает панели менеджера файлов. Для выхода из mc используется клавиша <F10> или последовательность клавиш <Esc>,<0>.

Помимо встроенного текстового редактора mc, в системе доступны также и другие текстовые редакторы. Для пользователей *nix-систем представляется полезным иметь хотя бы минимальные навыки работы с редактором vi — как стандартным редактором, имеющимся практически во всех системах.

При запуске редактора vi в командной строке ему указывается имя файла для редактирования:

```
vi file.txt
```

Если файл существует, то vi загружает его и отображает на экране, если нет — при сохранении создаётся новый файл с указанным именем. vi (*visual editor*) впервые появился в середине 70-х годов и имеет интерфейс, приспособленный для работы на самых простых терминалах. vi работает в двух основных режимах — в режиме «ввода текста» и в режиме «команд».

После запуска vi оказывается в режиме «команд». В этом режиме можно перемещаться по тексту с помощью клавиш управления курсором. На тех терминалах, где таких клавиш нет, можно использовать клавиши <h>, <l> (влево, вправо), <j>, <k> (вниз, вверх). Для перемещения курсора на следующий символ X в строке используется последовательность <f>,<X> (<f>,<"> - перемещение на символ "), в обратном направлении – <F>,<X> . Для перехода в начало строки используется команда <^>, в конец – <$> .

Для перехода на строку с номером N можно набрать её номер и команду <G> (<1>,<0>,<G> – переместить курсор на 10ую строку). Также можно осуществить поиск по тексту, нажав символ </> и введя нужный шаблон поиска. Повторный поиск в прямом и обратном направлении осуществляется клавишами <n> и <N> .

Найдя нужное место, можно перейти в режим «ввода текста». Для этого надо нажать <i> для вставки текста в текущей позиции, или <a> для добавления в конце строки. Для вставки новой строки в режиме ввода текста используется клавиша <Enter>. Выйти из режима ввода текста можно, нажав <Esc>.

В командном режиме можно удалять символы и строки текста. Для удаления символа под курсором используется клавиша <x>, для удаления строки — последовательность <d>,<d>. Удалённые символы или строки помещаются в буфер обмена. Содержимое буфера обмена можно вставить в текст клавишей <p>. Просто поместить текущую строку в буфер обмена можно последовательностью <y>,<y>.

Перед командой можно задать число её повторений. Например, последовательность клавиш <1>,<0>,<j> переместит курсор на 10 строк вниз, <7>,<l> - на 7 символов вправо, <8>,<x> удалит в буфер обмена 8 символов, начиная с текущей позиции, <d>,<5>,<k> - удалит в буфер обмена 5 строк вверх, начиная с текущей.

Для завершения редактирования файла и сохранения результата надо набрать в командном режиме :wq; для завершения редактирования без сохранения — :q!.

Возможности vi (и его улучшенной и расширенной версии vim) не ограничиваются простым редактированием текста, для их изучения можно использовать встроенное интерактивное руководство, вызываемое командой vimtutor.

Выйти из командного интерпретатора и завершить сеанс работы с системой можно, введя команду logout.

Следующие команды доступны администратору системы и позволяют управлять пользователями, их правами для доступа к файлам, и т.п.

$ su -l	Запустить командный интерпретатор с правами суперпользователя. Ключ -l обязателен. При выполнении команда запросит пароль суперпользователя. Завершить работу командного интерпретатора можно, как и в случае обычного пользователя, командной logout.

tcsh) имеют функции управления заданиями, позволяющие запускать одновременно несколько команд или заданий и, по мере надобности, переключаться между ними.

Управление заданиями может быть полезно, если, например, при редактировании большого текстового файла возникает необходимость временно прервать редактирование и выполнить какую-нибудь другую операцию. С помощью функций управления заданиями можно приостановить работу с редактором, вернуться к приглашению командной оболочки и запустить какие-либо другие команды. Когда они будут выполнены, можно вернуться обратно к работе с редактором в то его состояние, на котором была прервана работа с редактируемым файлом.

Передний план и фоновый режим.

Задания могут выполняться или на переднем плане (*англ.* foreground), или в фоновом режиме (*англ.* background). На переднем плане в любой момент времени может быть только одно задание. Задание на переднем плане взаимодействует с пользователем, получает ввод с клавиатуры терминала и посылает вывод на экран. Задания в фоновом режиме не получают ввода с терминала и обычно ничего на него не выводят (в противном случае выводящиеся из них данные будут произвольным образом смешиваться с выводом из команды переднего плана). Как правило, это задания, которые не нуждаются во взаимодействии с пользователем.

Некоторые задания исполняются очень долго, и во время их работы не происходит ничего интересного. Пример таких заданий — компилирование программ, а также сжатие больших файлов. Нет никаких причин смотреть на экран и ждать, когда эти задания выполнятся. Такие задания вполне можно запускать в фоновом режиме, тогда во время их выполнения будет возможность продолжать работать с системой.

Для управления выполнением процессов в Linux предусмотрен механизм передачи сигналов. Сигналы предоставляют процессам возможность обмениваться стандартными короткими сообщениями непосредственно с помощью операционной системы. Сообщение-сигнал не содержит никакой информации, кроме номера сигнала (для удобства вместо номера можно использовать предопределённое системой имя). Для того, чтобы передать сигнал, процессу достаточно задействовать системный вызов kill(). Для обработки поступающих сигналов процесс может зарегистрировать в системе для интересующих его сигналов свои процедуры-обработчики, или воспользоваться предоставляемыми системой стандартными обработчиками сигналов. В зависимости от номера сигнала стандартные обработчики или не выполняют никаких действий, или приводят к немедленному завершению получившего сигнал процесса.

Обработчик сигнала запускается асинхронно, немедленно после получения сигнала, что бы процесс в это время ни делал. В этом механизм сигналов очень похож на механизм обработки прерываний от аппаратной части компьютера; сигналы являются одним из вариантов внутренних прерываний в системе – так называемыми программными прерываниями.

Сигналы с номерами 9 (KILL) и 19 (STOP) всегда обрабатываются операционной системой. Первый из них принудительно останавливает и уничтожает процесс (отсюда и название, *англ.* kill — убивать). Сигнал STOP приостанавливает процесс: в таком состоянии процесс не удаляется из таблицы процессов, но и не выполняется до тех пор, пока не получит сигнал 18 (CONT), после чего продолжает работу. В командной оболочке Linux сигнал STOP можно передать активному процессу с помощью управляющей последовательности клавиш <Ctrl>+<Z>.

Сигнал номер 15 (TERM) служит для прекращения (*англ.* terminate) работы задания. При поступлении этого сигнала процесс должен завершить свою работу. Командная оболочка позволяет отправить сигнал TERM активному процессу с помощью управляющей последовательности <Ctrl>+<C>. При этом, в отличие от сигнала KILL, программы могут перехватывать сигнал TERM и установить собственный обработчик этого сигнала, т. е. нажатие комбинации клавиш <Ctrl>+<C> может и не прервать процесс немедленно. Это сделано для того, чтобы программа могла корректно завершить свою работу: удалить временные файлы, осуществить запись изменённых данных и т. п., прежде, чем она будет завершена. На практике, некоторые программы прервать таким способом не получится.

Существует утилита kill, предназначенная для отправления того или иного сигнала произвольному процессу. Её формат вызова:

```
kill [-s SIGNAL | -SIGNAL] PID
```

где SIGNAL — это посылаемый процессу сигнал, а PID — соответствующий идентификатор процесса. Например, для посылки сигнала KILL процессу 1 можно записать:

```
$ kill -9 1
-bash: kill: (1) - Операция не позволена
```

Запущенная обычным пользователем, такая команда закончится с ошибкой: на отправление сигналов также распространяются соглашения о контроле доступа, и обычный пользователь может отправлять сигналы только процессам, запущенным им самим (т. е. процессам с *UID* этого пользователя). Как говорилось в предыдущей лабораторной работе, процесс с *PID*, равным 1 — это процесс init, запускающийся первым после загрузки ядра операционной системы и от имени суперпользователя. Сам суперпользователь (администратор системы) может отправить любой сигнал любому процессу.

Перевод в фоновый режим и уничтожение заданий.

Рассмотрим управление заданиями на простого примере. Существует команда yes, которая выводит бесконечный поток строк, состоящих из символа y . При запуске этой команды на экран начинают выводиться строки с

цию клавиш, обычно это <Ctrl>+<Z>.

```
$ yes > /dev/null
Ctrl-Z[1]+  Stopped yes      >/dev/null
$
```

Приостановленный процесс попросту не выполняется, на него не тратятся вычислительные ресурсы процессора. Приостановленное задание можно вновь запустить на выполнение с той же точки, в которой оно было приостановлено, как будто бы этого не происходило.

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

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

Командная оболочка ещё раз выведет на экран название команды, чтобы пользователь знал, какое именно задание он в данный момент запустил на переднем плане. Приостановим это задание ещё раз нажатием клавиш <Ctrl>+<Z>, но в этот раз запустим его в фоновом режиме командой `bg` (от *англ.* background — фон). После перевода в фоновый режим процесс будет работать так, как если бы при его запуске использовалась команда с символом `&` (амперсанд) на конце (как это делалось в предыдущем разделе):

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

При этом приглашение командной оболочки возвращается пользователю, а команда `jobs` будет показывать, что процесс `yes` действительно в данный момент работает. Этот процесс можно уничтожить командой `kill`, как показывалось ранее.

Для того, чтобы приостановить работающее в фоновом режиме задание, нельзя воспользоваться комбинацией клавиш <Ctrl>+<Z>. Прежде, чем приостанавливать задание, его нужно перевести на передний план командой `fg`, и лишь потом приостановить. Таким образом, команду `fg` можно применять либо к приостановленным заданиям, либо к заданию, работающему в фоновом режиме. Другой вариант приостановки работающего в фоновом режиме задания – это отправка ему сигнала `STOP` командой `kill` .

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

```
$ yes &
```

Здесь стандартный вывод не был перенаправлен на устройство `/dev/null`, поэтому на экран будет выводиться бесконечный поток символов `y`. Этот поток невозможно будет остановить, поскольку комбинация клавиш

<Ctrl>+<C> не воздействует на задания в фоновом режиме. Для того чтобы остановить эту выдачу, надо использовать команду `fg`, которая переведёт задание на передний план, а затем уничтожить задание комбинацией клавиш <Ctrl>+<C>.

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

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

Отметим также, что функции управления заданиями реализуются средствами командного интерпретатора. Команды `fg`, `bg` и `jobs` являются внутренними командами оболочки, т. е. одноимённых файлов с их программным кодом в файловой системе нет. В простых командных интерпретаторах, например на встраиваемых системах, эти команды могут не поддерживаться. В этих случаях управлять работой процессов можно, посылая им сигналы стандартной командой `kill` .

Код возврата команд.

Любая запускаемая в системе команда (программа) выполняет какие-то действия, операции, задачи или успешно и без ошибок, или же в процессе работы программы возникают какие-либо проблемы, и выполнить поставленную задачу программа не может. О результатах своей работы и возникших ошибках программа сообщает запустившему её пользователю, выдавая текстовые информационные сообщения на экран. И, помимо этого, программа сообщает о результатах своей работы и операционной системе — через выдаваемый в операционную систему в момент своего завершения код возврата. Код возврата команды — это целое число, или равное нулю в случае успешного завершения команды, или не равное нулю в случае возникновения каких-либо ошибок. Возможные значения кодов возврата в случае ошибок выполнения команды зависят от конкретной команды и, как правило, приводятся на странице справочного руководства (`man`) по этой команде.

Код возврата последней выполненной команды командный интерпретатор запоминает в переменной `$?` (подробнее о переменных командного интерпретатора рассказывается ниже). Посмотреть его можно через команду `echo` :

```
$ ls /tmp
$ echo $?
```

Обычно команда cat читает данные из всех файлов, которые указаны в качестве её параметров, и посылает считанное непосредственно в стандартный вывод (*stdout*). Следовательно, команда

```
$ cat /etc/hosts /etc/resolv.conf
127.0.0.1 lab-00.edu.cbias.ru lab-00 localhost.localdomain localhost
192.168.212.250 ftp-distr
nameserver 192.168.212.252
```

выведет на экран сначала содержимое файла /etc/hosts, а затем — файла /etc/resolv.conf .

Однако если имя файла не указано, программа cat читает входные данные из *stdin* и немедленно возвращает их в *stdout* (никак не изменяя). Данные проходят через cat, как через «трубу». Приведём пример:

```
$ cat
Hello there.
Hello there.
Bye.
Bye.
Ctrl-D$
```

Каждую строчку, вводимую с клавиатуры, программа cat немедленно возвращает на экран. При вводе информации со стандартного ввода конец текста отмечается вводом специальной комбинации клавиш, как правило — <Ctrl>+<D>.

Приведём другой пример. Команда sort читает строки вводимого текста (также из *stdin*, если не указано ни одного имени файла) и выдаёт набор этих строк в упорядоченном виде в *stdout*. Проверим её действие.

```
$ sort
bananas
carrots
apples
Ctrl-D
apples
bananas
carrots $
```

Как видно, после нажатия <Ctrl>+<D> команда sort вывела строки упорядоченными в алфавитном порядке.

Перенаправление ввода и вывода.

Допустим, нужно направить вывод команды sort в некоторый файл, чтобы сохранить упорядоченный по алфавиту список на диске. Командная оболочка позволяет перенаправить стандартный вывод команды в файл, используя символ > (больше). Приведём пример:

```
$ sort > list
bananas
carrots
apples
```



```
Ctrl-D$
```

Можно увидеть, что результат работы команды sort не выводится на экран, однако он сохраняется в файле с именем list. Выведем на экран содержимое этого файла:

```
$ cat list
apples
bananas
carrots
$
```

Пусть теперь исходный неупорядоченный список находится в файле items. Этот список можно упорядочить с помощью команды sort, если указать ей, что она должна читать данные из этого файла, а не из своего стандартного ввода, и, кроме того, перенаправить стандартный вывод в файл, как это делалось выше. Пример:

```
$ sort items > list
$ cat list
apples
bananas
carrots
$
```

Однако можно поступить иначе, перенаправив не только стандартный вывод в файл, но и стандартный ввод утилиты из файла, используя для этого символ < (меньше):

```
$ sort < items
apples
bananas
carrots
$
```

Результат команды sort < items эквивалентен команде sort items, однако при выдаче команды sort < items система ведёт себя так, как если бы данные, которые содержатся в файле items, были введены со стандартного ввода. Перенаправление ввода-вывода осуществляется командной оболочкой. Команде sort не сообщалось имя файла items, эта команда читала данные из своего стандартного ввода, как если бы их вводили с клавиатуры.

Введём понятие фильтра. Фильтром является программа, которая читает данные из стандартного ввода, некоторым образом их обрабатывает и результат направляет в стандартный вывод. Когда применяется перенаправление, в качестве стандартного ввода и вывода могут выступать файлы. Как указывалось выше, по умолчанию, *stdin* и *stdout* относятся к клавиатуре и к экрану соответственно. Программа sort является простым фильтром: она сортирует входные данные и посылает результат на стандартный вывод. Совсем простым фильтром является программа cat: она ничего не делает с входными данными, а просто пересылает их на выход.

pnmtojpeg также выдаёт на стандартный выход, который средствами командного интерпретатора перенаправляется в файл /tmp/mc.jpg.

Другой пример: утилита mkisofs создаёт для файлов из заданного ей в качестве параметра каталога образ диска с файловой системой *ISO9660* для записи на оптические диски. А утилита cdrecord умеет записывать такие образы непосредственно на сами диски. Утилиты могут использоваться по-отдельности, с записью образа файловой системы в файл и последующей записью такого файла на диск. Однако их можно объединить в связку и записывать диски без создания временных файлов:

```
$ mkisofs ~/mydisk | cdrecord -
```

Здесь для того, чтобы приказать cdrecord использовать данные со стандартного входа, а не читать их из файла, мы в качестве имени файла указали – (дефис).

Недеструктивное перенаправление вывода и ввод до разделителя.

Эффект от использования символа > (больше) для перенаправления вывода в файл является деструктивным. Иными словами, команда

```
$ ls > file-list
```

уничтожит содержимое файла file-list, если этот файл ранее существовал, и создаст на его месте новый файл. Если вместо этого перенаправление будет сделано с помощью символов >>, то вывод будет дописан в конец указанного файла, при этом исходное содержимое файла не будет уничтожено. Например, команда

```
$ ls >> file-list
```

дописывает вывод команды ls в конец файла file-list.

Симметричная по виду запись перенаправления ввода (с помощью символов <<) используется для организации так называемого ввода до разделителя:

```
$ cat <<END
Hello, world!
END
Hello, world!
$
```

Здесь командный интерпретатор, встретив оператор перенаправления <<, запомнил последовательность символов после него (END) как разделитель потока ввода. Все последующие строки, вплоть до строки, содержащей только этот разделитель, были переданы на вход команды cat в виде потока ввода.

Следует иметь в виду, что перенаправление ввода и вывода и стыковка команд осуществляются командными оболочками, которые поддерживают использование символов >, >> , | и др. Сами команды специальным

образом эти символы не интерпретируют. Если нужно передать в команду один из этих символов в качестве параметра или использовать внутри передаваемой как параметр строки, то сделать это можно или «экранировав» одиночный спецсимвол с помощью символа обратного слеша (например, \<), или используя одинарные кавычки для выделения подстроки целиком.

Перенаправление потока вывода ошибок.

По-умолчанию операторы перенаправления > и >> изменяют передаваемый запускаемой программе файловый дескриптор с номером 1 – который соответствует потоку вывода. Возможно отдельно задать номер изменяемого файлового дескриптора, указав его перед операторами. Потоку вывода ошибок соответствует файловый дескриптор 2: т. е., например, для перенаправления вывода ошибок команды mkdir в /dev/null можно записать:

```
$ mkdir /etc/my-directory 2> /dev/null
```

Можно одновременно перенаправить и поток вывода, и поток ошибок:

```
$ ls -R /var/log/ 2>stderr >stdout
```

Здесь вывод команды ls -R (-R — рекурсивно по всем подкаталогам) выводится в файл stdout, а сообщения об ошибках – в файл stderr.

Также можно перенаправить стандартный поток ошибок в стандартный поток вывода – операторы перенаправления позволяют указать вместо имени файла номер файлового дескриптора в формате &номер:

```
$ ls -R /var/log/ 2>&1
```

При использовании одновременно перенаправления и стандартного потока вывода, и стандартного потока ошибок важен порядок операций:

```
$ ls -R /var/log/ 2>&1 >/dev/null
$ ls -R /var/log/ >/dev/null 2>&1
```

Первая команда присвоит файловому дескриптору потока ошибок значение файлового дескриптора потока вывода, и далее перенаправит поток вывода в /dev/null. В итоге сообщения об ошибках будут выводится в поток вывода (т. е. при запуске из терминала — на экран), а сам вывод команды будет перенаправляться в /dev/null.

Вторая команда сначала переопределит поток вывода, направив его в /dev/null, а потом присвоит потоку вывода ошибок значение файлового дескриптора потока вывода. В итоге, весь вывод команды – и стандартный, и сообщения об ошибках, - будет направлен в /dev/null.

	аааа или *ааааа*.

При использовании диапазонов символов следует учитывать, что они могут зависеть от выбранных настроек локализации. Например, диапазон «`[b-e]`» означает символы от *b* до *e* включительно. В английском языке, где сортировка букв идёт по-порядку (...*XYZabcdefg*...), ему соответствует набор символов *b*,*c*,*d*,*e*. По правилам русского языка, сортировка тех же символов идёт в другом порядке (...*эЭюЮяЯаAbBcCdDeEfFgG*...), и тому же диапазону соответствуют символы *b*,*B*,*c*,*C*,*d*,*D*,*e*.

Для решения таких проблем в стандарте *POSIX* имеются объявления некоторых классов и категорий символов:

Класс	Диапазон для английского языка	Описание
`[:upper:]`	`[A-Z]`	Латинские буквы верхнего регистра.
`[:lower:]`	`[a-z]`	Латинские буквы нижнего регистра.
`[:alpha:]`	`[A-Za-z]`	Латинские буквы верхнего и нижнего регистра.
`[:alnum:]`	`[A-Za-z0-9]`	Цифры, латинские буквы верхнего и нижнего регистра.
`[:digit:]`	`[0-9]`	Цифры.
`[:xdigit:]`	`[0-9A-Fa-f]`	Шестнадцатеричные цифры.
`[:punct:]`	`[.,!?:…]`	Знаки пунктуации.
`[:blank:]`	`[ \t]`	Пробел и табуляция.
`[:space:]`	`[ \t\n\r\f\v]`	Символы пропуска.
`[:cntrl:]`	-	Символы управления.
`[:graph:]`	`[^\t\n\r\f\v]`	Символы печати.

Способ представить сами метасимволы — `.`, `-` `[ ]` и другие — в регулярных выражениях без интерпретации, т.е. в качестве простых (не специальных) символов — предварить их («экранировать») символом `\` (обратный слеш). Например, чтобы представить сам символ «точка» (просто точка, и ничего более), надо написать `\.` (обратный слеш, а за ним — точка). Чтобы представить символ открывающей квадратной скобки `[`, надо написать `\[` (обратный слеш, и следом за ним скобка `[`) и т.д. Сам метасимвол `\` (обратный слеш) тоже может быть экранирован, то есть

представлен как `\\` (два обратных слеша), и тогда интерпретатор регулярных выражений воспримет его как простой символ обратного слеша `\`.

При составлении регулярных выражений следует также учитывать их две основные черты: они являются т.н. «ленивыми» и «жадными». Первое означает, что в строке, где есть несколько совпадений с шаблоном, шаблон совпадёт с первым из них. Например, регулярное выражение «*шаблон*`\(..\)`» для строки

в строке, где есть несколько совпадений с шаблоном, шаблон совпадёт с первым из них

вернёт в подвыражении `\1` символы *ом*, соответствующие первому встретившемуся подходящему совпадению (*шаблоном*). Второе возможное место совпадения (*шаблон с*) рассмотрено не будет.

«Жадность» регулярных выражений заключается в том, что, при использовании квантификаторов `*` (астериск) и `+` (плюс), шаблон будет совпадать с максимально длинным из возможных вариантов. Для той же строки шаблон «*шаблон*`.*`*н*», означающий подстроку, начинающуюся с «*шаблон*», заканчивающуюся на «*н*» и с произвольным количеством (`*`) любых (`.`) символов между «*шаблон*» и «*н*», совпадёт с подстрокой

шаблоном, шаблон совпадёт с первым из н,

а не с более короткой

шаблоном, шаблон

Рассмотрим далее применение регулярных выражений на примерах использования утилит `grep` и `sed`.

Утилита grep.

Одной из программ, использующих регулярные выражения для работы с текстом, является утилита `grep`. Она читает текст из файла и выводит те строки, которые совпадают с заданным регулярным выражением. Общий формат вызова утилиты:

```
grep [options] PATTERN [FILE...]
```

где `PATTERN` — регулярное выражение, а `FILE` — один или несколько файлов, к содержимому которых будет применено это регулярное выражение.

Если файл не задан, то `grep` читает текст со стандартного ввода. С помощью опций (*англ.* options) можно управлять поведением `grep`, например. опция `-v` приводит к выводу всех строк, не совпадающих с заданным регулярным выражением.

Итого, строкам комментариев соответствует выражение «^ *#», а пустым строкам — «^ *$». Как было отмечено ранее, фильтру grep можно приказать выводить строки, которые не совпадают с регулярным выражением, вызвав его с ключом -v.

Выводим файл lighttpd.conf в *stdout* и последовательно пропускаем вывод через два фильтра:

```
# cat /etc/lighttpd/lighttpd.conf | grep -v '^ *#' | grep -v '^ *$'
```

Этот вариант не очень эффективен, хотя и приносит желаемый результат. Можно избежать двух последовательных вызовов grep, объединив шаблоны. Видно, что они очень похожи: возможные пробелы в начале строки и или # (октоторп), или конец строки. Т.е. общий шаблон — «^ *(#|$)».

grep поддерживает несколько вариантов синтаксиса регулярных выражений и в варианте по умолчанию рассматривает круглые скобки как обычные символы. Поэтому надо или приказать grep'у рассматривать их как оператор выбора, экранировав скобки символом \ (обратный слеш), или переключить grep в режим работы с расширенным синтаксисом регулярных выражений, вызвав его с ключом -E, или использовать версию grep с включённой по умолчанию поддержкой расширенных регулярных выражений — egrep:

```
# cat /etc/lighttpd/lighttpd.conf | grep -v '^ *\(#\|$\)'
# cat /etc/lighttpd/lighttpd.conf | grep -E -v '^ *(#|$)'
# cat /etc/lighttpd/lighttpd.conf | egrep -v '^ *(#|$)'
```

Ну и наконец, нам не обязательно передавать файл lighttpd.conf на стандарный вход grep/egrep, эти утилиты могут сами прочитать файл с диска:

```
# egrep -v '^ *(#|$)' /etc/lighttpd/lighttpd.conf
```

Утилита sed.

Программа grep выполняет только поиск строк и выводит найденные результаты без изменений. Однако часто бывает необходимо не только найти какой-либо текст, но и изменить его. Для редактирования потока текста можно использовать утилиту sed (от *англ.* Stream EDitor, потоковый редактор). sed используется для выполнения основных преобразований текста, читаемого из файла или поступающего из стандартного потока ввода, и совершает одно действие над вводом за проход. Общий формат вызова sed:

```
sed [options] COMMAND [FILE...]
```

Из большого числа возможных команд sed мы рассмотрим только команду поиска и замены текста. Эта команда имеет вид s/PATTERN/EXPRESSION/ и осуществляет поиск в каждой из входящих строк текста регулярного

выражения PATTERN. Результаты совпадения заменяются на выражение EXPRESSION. Результирующий текст выводится в стандартный поток вывода.

Рассмотрим использование команды замены в sed на примерах.

В простейшем случае просто поменяем один фрагмент текста на другой:

```
$ ls -1 /var/cache
apt
fontconfig
man
$ ls /var/cache/ | sed 's/apt/APT/'
APT
fontconfig
man
```

В каталоге /var/cache есть несколько файлов, список их можно получить командной ls. Регулярное выражение «apt» совпадает с одной из строк вывода, и мы меняем совпадение на APT.

```
$ ls /var/cache/ | sed 's/a/A/'
Apt
fontconfig
mAn
```

В этом случае мы заменили в выводе ls букву *a* на *A*. sed применяет свои команды для каждой из строк вывода, поэтому в обеих строках, где была буква *a*, она была заменена.

Утилита uptime выдаёт определённую статистику по работе системы:

```
$ uptime
 07:48:42 up 27 days, 22:13,  1 user,  load average: 0.00, 0.00, 0.00
```

Для того, чтобы выделить из этой строки текущее число пользователей в системе, используем sed. Число пользователей — это одна или несколько цифр — «[0-9]\+», за которыми после пробела (или нескольких пробелов в общем случае) — «[0-9]\+ \+» следует слово *user* (или *users*). Нам интересно число пользователей — выберем его в подвыражении: «\([0-9]\+\) \+user». В начале строки идёт некоторый текст, отделённый от числа пользователей пробелом: «^.* \([0-9]\+\) \+user». Конец строки тоже может быть любой: «^.* \([0-9]\+\) \+user.*».

Данное выражение совпадает со всей строкой и выделяет в подстроку \1 число пользователей. Заменив целиком строку на \1, мы получим в результате только это число:

```
$ uptime | sed 's/^.* \([0-9]\+\) \+user.*/\1/'
1
```

Аналогично можно получить, например, время работы системы (подстроку вида *27 days, 22:13*):

```
$ ls -l /bin | awk '{print $9,$3":"$4,$1;}' | head
 : total
awk root:root lrwxrwxrwx
basename root:root -rwxr-xr-x
bash root:root -rwxr-xr-x
bash2 root:root lrwxrwxrwx
bunzip2 root:root lrwxrwxrwx
bzcat root:root lrwxrwxrwx
bzip2 root:root -rwxr-xr-x
bzip2recover root:root -rwxr-xr-x
cat root:root -rwxr-xr-x
```

Можно отфильтровать список и вывести только файлы. Для файлов первый символ поля прав — - (дефис). Для форматирования вывода разделим выводящиеся значения символами табуляции (код символа `\t`). С учётом этого получаем:

```
$ ls -l /bin | awk '/^-/ {print $9"\t->\t"$3":"$4"\t"$1;}' | head
basename        ->      root:root       -rwxr-xr-x
bash    ->      root:root       -rwxr-xr-x
bzip2   ->      root:root       -rwxr-xr-x
bzip2recover    ->      root:root       -rwxr-xr-x
cat     ->      root:root       -rwxr-xr-x
chgrp   ->      root:root       -rwxr-xr-x
chmod   ->      root:root       -rwxr-xr-x
chown   ->      root:root       -rwxr-xr-x
clock_unsynced  ->      root:root       -rwxr-xr-x
cp      ->      root:root       -rwxr-xr-x
```

Создание скриптов.

До сих пор нами рассматривался запуск программ из командной строки оболочки. Однако для повторяющихся последовательностей команд это неудобно. В таких случаях можно сохранить последовательность команд в файл и запускать их не из командной строки, а из такого файла. Обычно такие файлы с записанными командами называют скриптами.

В простейшем случае, скрипт можно создать, например, так:

```
$ echo "ls | grep script" > script
$ cat script
ls | grep script
$ sh script
script
```

Здесь мы создали текстовый файл, содержащий команды `ls` и `grep`, и далее выполнили эти команды, вызвав интерпретатор команд и передав ему в качестве аргумента имя скрипта. Интерпретатор команд, получив в качестве аргумента имя файла, считал из него команды и выполнил их.

Такой способ запуска скриптов не очень удобен. Он отличается от вызова команд системы: здесь требуется в командной строке указывать имя интерпретатора команд и, в общем случае, полный путь к выполняемому скрипту, в то время как для скомпилированных команд системы достаточно ввести имя самой команды. Кроме того, для операционных систем *nix

существует несколько альтернативных командных интерпретаторов с различным синтаксисом команд. Существует и большое количество различных интерпретирующих языков программирования, программы для которых также оформляются в виде скриптов и запускаются с помощью соответствующих программ-интерпретаторов. Таким образом, требуется способ указать системе, каким именно интерпретатором следует выполнять тот или иной скрипт.

Имя программы, которая должна интерпретировать записанную в текстовый файл (скрипт) последовательность команд, можно указать в самом скрипте. Это делается с помощью специальным образом оформленной первой строки скрипта, которая обычно выглядит примерно как

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

Первая строка состоит из двух символов `#!` (октоторп и восклицательный знак) и следующим за ними полном пути к программе, которая будет обрабатывать данный скрипт. В данном случае это интерпретатор команд `bash`. Как правило, интерпретируемые языки программирования (и командный интерпретатор в частности) используют символ # (октоторп) для выделения комментариев, т. е. интерпретировать подобным образом оформленную строку они не будут.

Как рассматривалось в предыдущей лабораторной работе, в операционных системах *nix существуют права доступа к файлам. Если для файла задано право его выполнения, то интерпретатор команд откроет его и прочитает несколько первых символов файла. Если там обнаружится начало скомпилированной программы, то она будет запущена, если же там обнаружится последовательность символов `#!`, то будет запущен указанный после неё интерпретатор, которому будет передано в качестве аргумента имя файла.

Итого:

```
$ echo '#!/bin/bash' > script
$ echo 'ls | grep script' >> script
$ chmod a+x script
$ cat script
#!/bin/bash
ls | grep script
$ ls -l script
-rwxr-xr-x 1 student student 29 Map 20 09:35 script
$ ./script
script
```

Здесь мы создали путём вызова двух команд `echo` файл (обратите внимание, что во второй команде мы дописали строку в имеющий файл), задали этому файлу право на выполнение, проверили результат (выведя файл через `cat` и проверив права на него через `ls -l`) и запустили его на выполнение.

Отметим, что командный интерпретатор ищет выполняемые файлы в определённых каталогах: `/bin`, `/usr/bin` и т.п. Для запуска программы из

обычного пользователя, и т.п.

Оператор присваивания.

Присвоение значений переменным осуществляется с помощью оператора = (знак равенства). Пробелов между именем переменной, = и значением быть не должно. Например:

```
$ A=5
$ B=пять
$ C=$A+$B
$ echo A
A
$ echo B=$B
B=пять
$ echo C=$C
C=5+пять
```

Как мы видим, интерпретатор команд все переменные рассматривает как строки. Однако есть возможность и вычисления арифметических выражений — через внешние программы.

Вычисление выражений.

Вычисление выражений осуществляется с помощью команды `expr` и арифметических и логических операторов:

```
$ a=5 b=12
$ a=`expr $a + 4`
$ d=`expr $b - $a`
$ echo $a $b $d $A
9 12 3 5
```

Для `expr` аргументы и операции обязательно разделяются пробелами (они должны передаться команде как отдельные параметры). Кроме того, мы видим, что имена переменных чувствительны к регистру, `a` и `A` — разные переменные.

Команда `expr` позволяет производить операции только над целочисленными значениями. Для выполнения вычислений с числами с фиксированной точностью или с вещественными значениями можно использовать другие команды (например, калькуляторы `dc` или `bc`) – хотя, в целом, язык *shell* не предназначен для решения вычислительных задач.

Условные выражения.

Ветвление вычислительного процесса осуществляется с помощью оператора `if`:

```
if список_команд1; then
     список_команд2
[else
     список_команд3]
fi
```


(В квадратных скобках указывается необязательная часть команды.)

`Список_команд` — это одна или несколько команд, для задания пустого списка используется : (двоеточие). `Список_команд1` передает оператору `if` код возврата последней команды из списка. Если код равен 0, то выполняются команды из `списка_команд2`, таким образом нулевой код возврата эквивалентен значению «истина». В противном случае выполняются команды из `списка_команд3`, если он указан.

Проверка условия может осуществляется с помощью команды `test`. Аргументами этой команды могут быть имена файлов, числовые и нечисловые строки. Она используется в следующих режимах:

- Проверка файлов: `test -ключ имя_файла`

  Ключи: `-r` файл существует и доступен для чтения;

  `-w` файл существует и доступен для записи;

  `-x` файл существует и доступен для исполнения;

  `-f` файл существует и является обычным файлом (т. е. не каталогом, не файлом устройства и т.п.);

  `-s` файл существует, является обычным файлом и не пуст, т. е. его размер больше 0 байт;

  `-d` файл существует и является каталогом.

- *Сравнение чисел:* `test число1 -ключ число2`

  Ключи: `-eq` равно;

  `-ne` не равно;

  `-lt` меньше;

  `-le` меньше или равно;

  `-gt` больше

  `-ge` больше или равно.

- Сравнение строк: `test [строка1] выражение строка2`

  `[-n] строка` строка не пуста;

  `-z строка` строка пуста;

  `строка1 = строка2` строки равны;

  `строка1 != строка2` строки не равны.

В качестве альтернативой записи `test` можно использовать команду `[` (открывающая квадратная скобка), при этом, например, для проверки существования файла вместо

Для скриптов на языке командного интерпретатора есть возможность указать оболочке автоматически прекратить работу скрипта при возникновении ошибки при выполнении любой команды. Данный режим включается командой '`set -e`' в начале скрипта.

Например, скрипт

```
#!/bin/bash
# Clear /root/tmp/log
cd /root/tmp
rm -r log/
mkdir log/
```

опасен — при запуске не от суперпользователя команда `cd` не сможет перейти в каталог `/root/tmp`, и может быть удалён каталог `log` со всем содержимым в текущем каталоге.

Можно явно проверить результат выполнение команды `cd /root/tmp` и завершить выполнение скрипта с выдачей кода ошибки:

```
#!/bin/bash
# Clear /root/tmp/log
cd /root/tmp || exit 1
rm -r log/
mkdir log/
```

Другой вариант – использовать '`set -e`':

```
#!/bin/bash
# Clear /root/tmp/log
set -e
cd /root/tmp
rm -r log/ ||:
mkdir log/
```

Здесь скрипт будет завершён автоматически при невозможности выполнить смену каталога. При этом, если в `/root/tmp` нет подкаталога `log/`, то команда рекурсивного удаления подкаталога также завершится в ошибкой. Чтобы при этом не было прервано выполнение скрипта, этот код ошибки надо обработать — в данном случае проигнорировать. Последовательность символов '`||:`' интерпретируется как оператор «или» и пустой оператор ':', всегда возвращающий нулевое значение.

Установка, удаление и обновление программных компонентов в системе.

Для операционной системы Linux доступно огромное количество программного обеспечения. Основная масса этого ПО доступна под свободными лицензиями, и с сайтов разработчиков можно загрузить архивы с исходными текстами программ. Однако пригодные для запуска скомпилированные бинарные файлы разработчиками программ или предоставляются для очень ограниченного круга дистрибутивов, или не предоставляются совсем.

Хотя обычно процесс сборки программ из исходных текстов автоматизирован, включая нахождение необходимых библиотек и заголовочных файлов на конкретной системе, всё-таки данное занятие требует достаточно глубоких знаний. При наличии нескольких систем собирать и устанавливать программу придётся вручную на каждой по-отдельности, т.к. набор и версии установленных системных библиотек на разных системах могут отличаться. Кроме того, перед использованием программу мало собрать — её надо установить в соответствующие данному дистрибутиву каталоги, внести изменения в её настройки (и, возможно, в настройки других программ) в соответствии с принятыми в конкретном дистрибутиве соглашениями, убедиться в правильности прав на установленные файлы, при необходимости создать псевдопользователей, добавить скрипты для запуска программы и т.п. Дополнительные проблемы возникают при появлении новых версий установленного на системе программного обеспечения — так, при обновлении какой-либо системной библиотеки, требуется заново собрать и установить не только её, но и все использующие её программы.

Для решения этой проблемы были разработаны системы, позволяющие компилировать программы и распространять результат в виде пакетов — архивов специального формата. В заголовке пакета указывается информация о названии программы, краткое её описание, номер версии программы и версии самого пакета. Для каждого пакета указываются его зависимости от других пакетов — т.е. пакеты с теми программами и библиотеками, которые используются программой в пакете и нужны ей для работы.

На данный момент существуют и широко используются две системы сборки пакетов программ. Первая — *Debian package management system (dpkg)*, использующая для установки и обновления пакетов программу `dpkg`, работающую с форматом `.deb` . Данная система используется в проекте Debian и вышедших из него дистрибутивах семейства Ubuntu.

Вторая система — *RPM Package Manager* (изначально *Red Hat Package Manager*), разработанная компанией Red Hat. В этой системе для управления пакетами в формате *RPM* используется одноимённая утилита. Пакеты в формате *RPM*, в частности, используются в дистрибутивах Red Hat, SUSE, Mandriva, в проектах Fedora Core, PLD, в отечественных проектах ASP Linux и ALT Linux.

Пакеты разделяются на две категории — пакеты с исходными текстами и пакеты с исполняемым кодом (бинарные пакеты). В первых содержится исходный код программы и инструкции для системы управления пакетами по сборке из этого исходного кода пакетов с исполняемым кодом. В системе *RPM* такие пакеты имеют расширение `.src.rpm`. Пакеты с исполняемым кодом содержат скомпилированные программы и предназначены для установки этих программ в системе.

Исполняемый код, очевидно, зависит от архитектуры системы. Одному пакету с исходным кодом, таким образом, соответствует несколько пакетов с двоичным. На данный момент в составе ALT Linux поддерживаются архитектуры 32-битных процессоров с командами Intel Pentium (с расширением

Для уже выпущенных версий дистрибутивов изменения в их репозитории не вносятся, а новые версии программ, в т.ч. с исправлением выявленных ошибок, включаются в отдельные репозитории обновлений.

Одной из систем, позволяющих работать с репозиториями пакетов, является *APT* (*Advanced packaging tool*). Изначально разработанная для *dpkg*, в настоящее время она также может работать и с репозиториями пакетов *RPM*.

Перед использованием системы *APT* ей следует указать, какие репозитории она должна использовать. Эти настройки хранятся в каталоге `/etc/apt/`, в файле `/etc/apt/sources.list` и в файлах в каталоге `/etc/apt/sources.list.d/`. Запись о репозитории выглядит следующим образом:

```
 rpm [p9] ftp://ftp.altlinux.org/pub/distributions/ALTLinux/p9/branch x86_64
classic
 rpm [p9] ftp://ftp.altlinux.org/pub/distributions/ALTLinux/p9/branch noarch
classic
```

`rpm` указывает на тип репозитория. Для систем ALT Linux других значений в этом поле быть не должно. В квадратных скобках указывается имя открытого ключа, которым подписан репозиторий. Если цифровая подпись репозитория не будет соответствовать ключу, *APT* откажется работать с таким репозиторием. Далее указан URL самого репозитория. Как видно, в данном случае репозиторий доступен по протоколу FTP и размещён на сервере ftp.altlinux.org в каталоге `/pub/distribution/ALTLinux/p9/branch` . В четвёртом поле указывается архитектура пакетов в репозитории. В данном примере используются пакеты для 64-битных систем и архитектурно-независимые пакеты. Для 32-битных систем вместо `x64_86` должно указываться `i586`, для систем на архитектуре Эльбрус — `e2kv3` или `e2kv4`, и т.п. Последнее поле — используемый набор пакетов в репозитории.

Строки, начинающие с `#` — комментарии. Т.е. указанные в них репозитории не используются. Если их надо подключить, то символ `#` (октоторп) следует удалить.

В системе обычно используются удалённые репозитории, размещённые где-либо в Internet. Хотя можно разместить репозиторий локально в самой системе (тогда URL с путями к нему будут начинаться с file:///), чаще всего это нецелесообразно. Репозитории занимают довольно много места, причём значительная часть файлов в них для конкретной системы не нужна: в репозитории содержатся как пакеты с исходными текстами программ для самостоятельной сборки, так и пакеты для разных архитектур, из которых требуется только одна. Например, по состоянию на сентябрь 2018 г. приведённый выше репозиторий занимал порядка 270 Gb, из них около 60 Gb занимали пакеты с исходными кодами, по 40 Gb — пакеты для архитектур ARMv7 и MIPS, по 45 Gb — для x86_64, i586, 30 Gb – архитектурно-независимые пакеты. Кроме того, пакеты в репозитории постоянно обновляются (где-то 5-10 Gb в неделю для указанного репозитория). На конкретной же системе установлены только некоторые из пакетов в репозитории, и регулярно скачивать из Internet все изменения просто не нужно. Система *APT* поддерживает работу с удалёнными репозиториями и позволяет минимизировать сетевой трафик.

Для того, чтобы система *APT* узнала текущее состояние репозитория и список доступных пакетов в нём, требуется обновить её локальный кэш списка пакетов. Это делается командной `apt-get update`. В случае, если какие-либо репозитории недоступны, будут выведены сообщения об ошибках. Примерный вид работы `apt-get update` выглядит следующим образом:

```
# apt-get update
Get:1 ftp://ftp-distr x86_64 release [730B]
Get:2 ftp://ftp-distr noarch release [728B]
Fetched 1458B in 0s (13.5kB/s)
Get:1 ftp://ftp-distr x86_64/classic pkglist [2081kB]
Hit ftp://ftp-distr x86_64/classic release
Get:2 ftp://ftp-distr noarch/classic pkglist [942kB]
Hit ftp://ftp-distr noarch/classic release
Fetched 3023kB in 1s (1906kB/s)
Reading Package Lists... Done
Building Dependency Tree... Done
```

В данном случае система *APT* успешно обновила список пакетов.

Стоит обратить внимание на то, что выполнение операций по установке и обновлению пакетов в системе — это задача системного администратора. Поэтому `apt-get` должна вызываться с привилегиями суперпользователя.

Для обновления уже установленных пакетов в системе используется команда `apt-get upgrade`:

```
# apt-get upgrade
Reading Package Lists... Done
Building Dependency Tree... Done
The following packages will be upgraded
  libpcre3 shadow-convert shadow-utils startup
The following packages have been kept back
  pam0_passwdqc
4 upgraded, 0 newly installed, 0 removed and 1 not upgraded.
Need to get 564kB of archives.
After unpacking 4266B of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 ftp://ftp-distr x86_64/classic shadow-convert 1:4.0.4.1-alt8 [53.6kB]
Get:2 ftp://ftp-distr x86_64/classic shadow-utils 1:4.0.4.1-alt8 [351kB]
Get:3 ftp://ftp-distr noarch/classic startup 0.9.8.18-alt1 [33.6kB]
Get:4 ftp://ftp-distr x86_64/classic libpcre3 7.6-alt1 [126kB]
Fetched 564kB in 0s (2385kB/s)
Committing changes...
Preparing...        ########################################## [100%]
1: shadow-convert   ########################################## [ 25%]
....
4: libpcre3         ########################################## [100%]
Done.
```

При выполнении установки и обновления *APT* получает из Internet необходимые пакеты. Эти пакеты сохраняются в локальном кэше. Для сохранения места на диске данный кэш можно очистить командой `apt-get clean`.

Для поиска нужного пакета в репозитории используется команда `apt-cache search`. Указав ей в качестве параметров нужное имя программы или её описание, можно получить список пакетов. Например, на запрос о пакете с web-сервером выдаётся примерно такой список:

```
$ apt-cache search webserver
apache - Самый популярный веб-сервер Internet
apache-mod_perl - Веб-сервер Russian Apache со встроенным интерпретатором Perl
apache2 - Самый популярный веб-сервер Internet
dvdrip - DVD ripping graphical tool using transcode
furl - Display the HTTP headers returned by webservers
httpd-alterator - Apache HTTP Server (alterator edition)
lighttpd - A fast webserver with minimal memory-footprint
lighttpd-rrdtool - rrdtool support lighttpd module
mod_dav - Модуль DAV под Apache 1.3.x
php5-dbase - dBase database file access functions
freevo - Freevo
jetty5 - The Jetty Webserver and Servlet Container
mailgraph-common - Simple mail statistics for Postfix
maven-plugin-webserver - Optional webserver plugin for maven
perl-LWPx-ParanoidAgent - subclass of LWP::UserAgent that protects you from
harm
```

В репозиториях дистрибутивов содержится огромное количество программ. Если появляется необходимость установить какую-либо новую программу, её прежде всего стоит поискать в готовом виде в репозитории. Чаще всего найти её удастся. Программы, отсутствующие в репозиториях, не стоит собирать из исходных кодов и ставить в систему напрямую, без создания пакета, поскольку в этом случае сильно затрудняется дальнейшее сопровождение системы. Также не стоит особенно переживать, если на сайте разработчика есть более новая версия программы, чем та, что доступна в репозитории. Как правило, в таких случаях у собирающего пакет администратора есть какие-либо причины не обновлять версию в пакете.

Дополнительно можно отметить, что в случае установки программ из готовых пакетов на рабочих системах не требуется наличие компиляторов, заголовочных файлов и прочих инструментов, применяемых при разработке и сборке программ. Это, с одной стороны, позволяет уменьшить место, занимаемое системой на диске, а с другой — создать дополнительные сложности потенциальным злоумышленникам, часто собирающим необходимые им для взлома систем программы непосредственно на этих системах.

Запуск и остановка сервисов, настройка их автоматического запуска при загрузке системы.

Как правило, неинтерактивные программы — демоны или сервисы — должны запускаться при загрузке системы и корректно останавливаться при её выключении/перезагрузке. Для этого в *nix-системах используется система инициализации, в состав которой входит процесс `init`. Как говорилось ранее, процесс `init` запускается ядром операционной системы при загрузке системы. Далее этот процесс, согласно настройкам системы инициализации, запускает другие процессы, выполняющие настройку оборудования, проверку и монтирование файловых систем, запуск демонов, и т.д.

Традиционной и достаточно широко распространённой в настоящее время системой инициализации является система *sysvinit*, представляющая собой достаточно сложную систему скриптов. Для описания состояния системы в *sysvinit* вводится понятия уровня загрузки (уровня выполнения). В любой момент времени система находится на некотором определённом уровне загрузки, и в ней выполняется соответствующий этому уровню набор сервисов. Имеется возможность отдать системе команду и перевести её с текущего уровня на другой. Управляет переключениями уровней загрузки процесс `init`. При переходе с одного уровня на другой `init` последовательно запускает скрипты, останавливающие работающие на текущем уровне загрузки системы сервисы, и затем запускает сервисы, которые должны работать на новом уровне.

Уровней выполнения 7, из них:

- 0 — уровень остановки системы. На этот уровень система переходит по командам `poweroff`, `shutdown`, `halt`. Если подобное поддерживает аппаратная платформа, то после перехода на этот уровень компьютер выключается.
- 1 — однопользовательская система. Используется только в режиме восстановления системы, обычно на этом уровне запускается только командный интерпретатор суперпользователя.
- 2 — многопользовательская система без сетевой поддержки. Как правило, в настоящее время этот уровень не используется.
- 3 — многопользовательская система. Это основной уровень работы системы, используемый по умолчанию.
- 4 — предоставлено для настройки конкретных систем. Обычно то же самое, что и уровень 3, и не используется.
- 5 — многопользовательская система с поддержкой графики. Изначально для настольных системы предусматривалась загрузка на 3-ий уровень, если не требовался запуск графической среды, и на 5-ый — если графическая среда использовалась. В настоящий момент в настольных системах графическая среда запускается и на 3-ем уровне, т.е. 5-ый уровень практически не используется.

выше *sysvinit* отличается малой требовательностью к ресурсам для своей работы, и широко используется для серверных и встраиваемых систем. Основными её недостатками являются последовательное выполнение операций запуска или остановки демонов в процессе перехода с одного уровня выполнения на другой, и сложность задания правильной последовательности запуска и остановки зависящих друг от друга демонов.

Для современных настольных и серверных Linux-систем в настоящее время преимущественно выбирается система инициализации *systemd*, обеспечивающая параллельный запуск демонов при загрузке системы и, тем самым, существенно уменьшающая время загрузки. Для просмотра и управления конфигурацией в *systemd* используются консольная команда `systemctl` и её графический аналог `systemadm`.

В отличие от *sysvinit* в системе инициализации *systemd* для конфигурации сервисов используются не наборы скриптов на языке командного интерпретатора, а файлы конфигурации, описывающие порядок и параметры запуска сервисов. Вместо уровней выполнения используется понятие целей (*target*), для достижения нужной цели *systemd* определяет по файлам конфигурации нужный порядок остановки или запуска сервисов и выполняет соответствующие операции. Цель по-умолчанию носит название «`multi-user.target`».

Настройка и получение информации о работе сервисов выполняются с использованием команды `systemctl`. Для запуска сервиса в режиме командной строки используется команда `systemctl start <имя сервиса>` , для остановки - `systemctl stop <имя сервиса>` , для перезапуска - `systemctl restart <имя сервиса>`

Включить автоматический запуск сервиса используется можно командой `systemctl enable <имя сервиса>` , выключить автоматический запуск - `systemctl disable <имя сервиса>` . Также можно получить информацию о состоянии сервиса — `systemctl status <имя сервиса>` . Например,

```
 # systemctl enable lighttpd
 Synchronizing state of lighttpd.service with SysV service script with
/lib/systemd/systemd-sysv-install.
 Executing: /lib/systemd/systemd-sysv-install enable lighttpd
 Created symlink /etc/systemd/system/multi-user.target.wants/lighttpd.service →
/lib/systemd/system/lighttpd.service.

 # systemctl start lighttpd

 # systemctl status lighttpd
 ● lighttpd.service - Lighttpd Daemon
    Loaded: loaded (/lib/systemd/system/lighttpd.service; enabled; vendor
preset: disabled)
    Active: active (running) since Thu 2019-10-17 03:59:54 UTC; 2s ago
   Process: 32096 ExecStartPre=/usr/sbin/lighttpd -tt -f
/etc/lighttpd/lighttpd.conf (code=exited, status=0/SUCCESS)
  Main PID: 32097 (lighttpd)
     Tasks: 1 (limit: 4915)
    Memory: 968.0K
    CGroup: /system.slice/lighttpd.service
            └─32097

 Oct 17 03:59:54 lab-00.edu.cbias.ru systemd[1]: Starting Lighttpd Daemon...
 Oct 17 03:59:54 lab-00.edu.cbias.ru systemd[1]: Started Lighttpd Daemon.
```


В отличие от *sysvinit*, *systemd* может отслеживать выполнение нужных сервисов и в случае каких-либо сбоев автоматически перезапускать их.

Также в рамках *systemd* имеется централизованный сервис сбора и хранения журналов работы системы и сервисов — *journald*. Получить логи работы системы и сервисов можно, используя команду `journalctl`. По-умолчанию `journalctl` выводит все логи с начала последнего запуска систем, можно ограничить его вывод последними N строками (команда вида `journalctl -n 100`), или посмотреть логи запуска и выполнения конкретного сервиса (`journalctl -u <имя сервиса>`):

```
 # journalctl -u lighttpd -n 5
 -- Logs begin at Wed 2019-10-16 16:59:50 UTC, end at Thu 2019-10-17 04:01:11
UTC. --
 Oct 17 03:59:46 lab-00.edu.cbias.ru systemd[1]:
/lib/systemd/system/lighttpd.service:7: PIDFile= references a path below legacy
directory /var/run/, updating /var/run/lighttpd.pid → /run/lighttpd.pid; please
update the unit file accordingly.
 Oct 17 03:59:47 lab-00.edu.cbias.ru systemd[1]:
/lib/systemd/system/lighttpd.service:7: PIDFile= references a path below legacy
directory /var/run/, updating /var/run/lighttpd.pid → /run/lighttpd.pid; please
update the unit file accordingly.
 Oct 17 03:59:52 lab-00.edu.cbias.ru systemd[1]:
/lib/systemd/system/lighttpd.service:7: PIDFile= references a path below legacy
directory /var/run/, updating /var/run/lighttpd.pid → /run/lighttpd.pid; please
update the unit file accordingly.
 Oct 17 03:59:54 lab-00.edu.cbias.ru systemd[1]: Starting Lighttpd Daemon...
 Oct 17 03:59:54 lab-00.edu.cbias.ru systemd[1]: Started Lighttpd Daemon.ervice
lighttpd
```

Другие варианты использования `journalctl` можно посмотреть в его справочном руководстве man.

Нужно отметить, что часть программ не использует системные сервисы журналов, и самостоятельно записывают журналы работы в свои подкаталоги внутри каталога `/var/log/` .

В целях совместимости в системе инициализации systemd поддерживаются также скрипты *sysvinit* в `/etc/rc.d/init.d/` и команды `service` и `chkconfig`.

В используемом в настоящей лабораторной работе дистрибутиве ALT Linux Server P9 в качестве системы инициализации используется *systemd*, в рамках выполнения лабораторной работы можно использовать как команды `service` и `chkconfig` в режиме совместимости с *sysvinit*, так и напрямую команду `systemctl` .

Выполнение лабораторной работы.

Лабораторная работа посвящена изучению основ работы с операционной системой семейства *nix, основ взаимодействия команд в операционных системах семейства *nix, использованию перенаправления потоков ввода-вывода, регулярных выражений, написанию простых программ на языке командного интерпретатора и выполнению основных задач по администрированию операционной системы. Выполнение лабораторной работы предусматривает работу с удалённым сервером. Для доступа к серверу используется терминальная программа *PuTTY*.

В лабораторной работе требуется:

- ознакомится с основами работы в операционной системе ALT Linux Server, изучить работу основных команд операционной системы;
- провести начальную настройку и подготовку операционной системы к использованию;
- организовать периодическое получение данных о работе определён-ных систем;
- записывать их в файл для последующего анализа;
- организовать получение текущих значений через веб-интерфейс;
- построить графики изменения наблюдаемых величин и предоставить к ним доступ через веб-интерфейс.

Поскольку, как правило, под решение практически любой задачи в Linux можно найти в Internet или готовое решение, или набор рецептов, то выполнение лабораторной работы предусматривает использование готовых скриптов для выполнения поставленных задач. С другой стороны, данные скрипты надо установить на конкретную систему, адаптировать их под задачу и обеспечить их выполнение в рамках выделенного виртуального сервера.

Перед началом выполнения работы необходимо получить у преподавателя индивидуальные данные, содержащие:

- учётную запись пользователя на удалённом сервере: имя (идентификатор пользователя) и пароль;
- сетевой адрес сервера и номер порта для удалённого входа на него;
- имя сервера для доступа по протоколу http;
- список репозиториев для настройки системы *APT*.

В ходе данной лабораторной работы Вы должны изменить идентификатор пользователя и пароль для доступа к серверу. Остальные данные остаются постоянными для последующих работ в рамках данного курса.

Для входа на сервер требуется загрузить терминальную программу *PuTTY*. Внутри сети МЭИ на период проведения данного курса она доступна со страницы http://edu.cbias.ru/ или по прямой ссылке

http://edu.cbias.ru/files/putty.exe.

После запуска программы появляется окно с настройкой параметров соединения. Полученные сетевой адрес сервера и номер порта следует ввести в поля *Host Name (or IP address)* и *Port* соответственно. Для корректной работы с разными кодировками сервера и клиентской системы, требуется указать кодировку поступающих от сервера символов. Эти настройки задаются на вкладке *Windows → Translation*, выбираемой из списка слева в окне настроек. В выпадающем списке *Received data assumed to be in which character set:* требуется задать нужную кодировку (в данном случае - UTF-8), для старых версий *PuTTY* вместо *KOI8-U* следует указать *UTF-8*.

Для подключения к серверу и начала сеанса нажмите кнопку *Open* внизу окна настроек.

В появившемся окне консоли на запрос *login as:* введите идентификатор пользователя, на запрос *password* — пароль. Пароль при вводе не отображается. В случае ошибки повторите ввод пароля или перезапустите *PuTTY*.

После успешного входа в систему в окне терминала появляется приглашение вида:

```
[student@lab-100 ~]$
```

Дальнейшие команды, вводимые в терминале, выполняются на удалённом сервере.

Изучите структуру каталогов сервера, пользуясь командами `ls` (в т.ч. с ключами `-l`, `-la`, `-a`), `cd`, `pwd` .

Запустите менеджер файлов *Midnight Commander* (команда `mc`). Для перехода из оконного режима `mc` в консольный и обратно используйте сочетание клавиш `<Ctrl>+<o>`. Используйте `mc` для копирования, перемещения и удаления файлов. Повторите те же операции из командной строки, используя `cp`, `mv`, `rm`. Используйте возможности командного интерпретатора по автоматическому дополнению имени файлов при нажатии клавиши `<Tab>`.

Перейдите в каталог `~/Documents`, создайте пустой файл командой `touch`. Для получения справки по параметрам команды используйте команду `man`.

Для выхода из справочного руководства используйте клавишу `<q>`.

Введите какой-либо текст в созданный файл, используя встроенный редактор `mc` (`<F4>`).

Выйдите из *Midnight Commander*.

используется порт 80, для схемы https:// - 443, и, как правило, вместо адреса IP в браузере указывается доменное имя, соответствующее нужному адресу IP. (Подробнее вопросы работы протоколов TCP/IP рассматриваются в лабораторной работе № 3.)

Чтобы браузер мог подключиться к веб-серверу по адресе IP и порту TCP, веб-сервер должен ожидать входящие подключения на этот адрес IP и порт TCP. Как правило, на сервере есть несколько адресов IP на разных интерфейсах, и по каким из этих адресов веб-сервер ожидает подключение, указывается в его конфигурации.

В конфигурации lighttpd адреса, подключения по которым ожидает lighttpd, задаются в параметре конфигурации `server.bind`. По-умолчанию lighttpd принимает соединения только на локальный адрес сервера («`localhost`»).

Чтобы можно было подключиться к веб-серверу из любых внешних сетей, в данном параметре требуется задать значение «`0.0.0.0`».

Запустите веб-сервер `lighttpd`, выполнив команду `service lighttpd start`. Проверьте, работает ли сервер, выполнив команду `service lighttpd status`. Проверьте наличие сервера в списке выполняемых процессов, выполнив команду `ps aux`. Обратите внимание на пользователя, под которым выполняется процесс веб-сервера. Получите файл из браузера, указав имя сервера и имя файла.

Получите список зарегистрированных в системе сервисов, командой `chkconfig --list`. Убедитесь в присутствии в списке `lighttpd`. В случае его отсутствия, добавьте сервис командой `chkconfig lighttpd --add`. Для автоматического запуска при загрузке системы сервис должен быть включён на уровнях выполнения 2-5. Если он выключен, включите его командой `chkconfig lighttpd on`.

Перезапустите систему командой `reboot`. Дождитесь загрузки сервера (время перезагрузки находится в пределах 5 минут). Войдите в систему. Проверьте, работает ли `lighttpd`.

Проведя таким образом начальную настройку операционной системы, можно приступать к установке набора скриптов для сбора и отображения данных о работе системы, что является основной целью данной лабораторной работы.

Поскольку, как правило, под решение практически любой задачи в Linux можно найти в Internet или готовое решение, или набор рецептов, то выполнение лабораторной работы предусматривает использование готовых скриптов для выполнения поставленных задач. С другой стороны, данные

скрипты надо установить на конкретную систему, адаптировать их под задачу и обеспечить их выполнение в рамках выделенного виртуального сервера.

В лабораторной работе требуется получить, записать и проанализировать следующие значения:

- число процессов в системе. Данный параметр может быть получен путём вывода полного списка выполняющихся в системе процессов и подсчёта числа строк в этом списке.
- суммарный объем переданных и принятых через сетевой интерфейс `venet0` данных в байтах. Эти значения содержатся в выводе команды `netstat -i`, в соответствующих полях выдаваемой таблицы.
- Число переданных и принятых через порт удалённого коммутатора пакетов и байтов данных. Данные величины могут быть получены по протоколу *SNMP* с использованием программы `snmpget`.

Вызов программы snmpget имеет вид:

```
$ snmpget -c public -v 1 192.168.250.1 IF-MIB::ifDescr.2 \
> IF-MIB::ifInOctets.2 \
> IF-MIB::ifInUcastPkts.2 \
> IF-MIB::ifOutOctets.2 \
> IF-MIB::ifOutUcastPkts.2
IF-MIB::ifDescr.2 = STRING: eth0
IF-MIB::ifInOctets.2 = Counter32: 120684456
IF-MIB::ifInUcastPkts.2 = Counter32: 1215812
IF-MIB::ifOutOctets.2 = Counter32: 1559547791
IF-MIB::ifOutUcastPkts.2 = Counter32: 1341129
```

Здесь было произведено обращение к коммутатору `192.168.250.1`, с которого были запрошены параметры:

`IF-MIB::ifDescr.2` — имя 2-го сетевого интерфейса;
`IF-MIB::ifInOctets.2` — число принятых интерфейсом байтов;
`IF-MIB::ifInUcastPkts.2` — число принятых интерфейсом пакетов;
`IF-MIB::ifOutOctets.2` — число переданных интерфейсом байтов;
`IF-MIB::ifOutUcastPkts.2` — число переданных интерфейсом пакетов.

Вывод команды приведён выше.

IP-адрес коммутатора и номер сетевого интерфейса индивидуальны для каждого виртуального сервера, их можно найти в файле `/root/SNMP.data` . В случае отсутствия такого файла в системе эти данные должны быть получены у преподавателя.

В ходе лабораторной работы используются программы `netstat` и `snmpget`, которые можно установить из пакетов `net-snmp-clients` и `net-tools`. Для отображения графиков используется набор утилит `RRDTOOLS` из пакета `rrd-utils`.

Для вывода данных по запросам браузера предлагается установить в систему для запуска с помощью lighttpd следующие скрипты:

cgi-local.sh — отображение локальной статистики.

```
#!/bin/bash
# Simple CGI script

echo Content-type: text/plain
echo ""

LOG_FILE=/var/www/stat/local.log

# Show NUM lines
if [ -n "$QUERY_STRING" ]; then
    NUM=$QUERY_STRING
else
    if [ -n "$1" ]; then
        NUM=$1
    else
     NUM=10
    fi
fi

echo "Current statistic:"
tail -n $NUM "$LOG_FILE" | sort -r
#--------------------------------------------------------------
```

cgi-snmp.sh — отображение SNMP-статистики.

```
#!/bin/bash
# CGI script for SNMP statistic

echo Content-type: text/plain
echo ""

LOG_FILE=/var/www/stat/snmp.log

# Show NUM lines
if [ -n "$QUERY_STRING" ]; then
    NUM=$QUERY_STRING
else
    if [ -n "$1" ]; then
        NUM=$1
    else
     NUM=10
    fi
fi

echo "Current statistic:"
tail -n $NUM "$LOG_FILE" | sort -r
#--------------------------------------------------------------
```

Текст скриптов, обеспечивающих вывод данных в табличной форме и построение графиков, приведён на на странице с примерами к данной лабораторной работе – http://lab-00.edu.cbias.ru/ .

Скрипты предполагается размещать в каталогах внутри /var/www, с использованием для скриптов получения данных каталога /var/www/bin, для веб-интерфейса — document_root веб-сервера, для хранения журналов — /var/www/stat. Для хранения графиков используется подкаталог внутри document_root веб-сервера.

Для запуска скриптов как веб-программ следует разрешить это в настройках lighttpd (расположенных в каталоге /etc/lighttpd/):

- нужно подключить модуль mod_cgi веб-сервера, раскомментировав строку «include "conf.d/cgi.conf"» в файле modules.conf;
- задать в подключеном файле конфигурации модуля mod_cgi (conf.d/cgi.conf) секцию параметров вида:

```
    cgi.assign       = ( ".pl"  => "/usr/bin/perl",
                                    …
                         ".rrd" => "/usr/bin/rrdcgi",
                         ".sh" => "/bin/bash" )
```

- добавить расширения файлов скриптов в параметр static-file.exclude-extensions в файле lighttpd.conf:

```
    static-file.exclude-extensions = ( ".php", ".pl", ".fcgi", ".sh", ".rrd" )
```

Изменения настроек вступают в силу после перезапуска lighttpd.

Также для запуска скриптов на выполнение веб-сервер lighttpd должен иметь права на чтение использования каталога с ними.

Для обеспечения безопасности системы должны соблюдаться определённые правила выполнения скриптов.

Сбор статистики должен выполняться от имени непривилегированного пользователя. Обычно для подобных задач создаётся отдельный псевдо-пользователь с ограниченными по сравнению с обычными пользователями системы правами. Псевдопользователь не должен иметь возможности удалённого входа в систему и не должен иметь возможности изменения скриптов.

Отображающие информацию скрипты выполняются веб-сервером. Пользователь, под которым работает веб-сервер, не должен иметь возможности записи как в файлы скриптов, так и в файлы с сохранённой статистикой (файлы логов).

Временные файлы, создаваемые веб-сервером, не должны быть доступны для записи или удаления остальным пользователям системы.

Остальные пользователи системы не должны иметь возможности чтения и записи файлов логов.

Контрольные вопросы.

1. Какие основные каталоги есть в файловой системе *nix?
2. В каких каталогах хранятся настройки системы?
3. В каких каталогах можно найти установленные в системе программы, доступные для пользователя?
4. В каких каталогах можно найти установленные системные программы и программы, предназначенные для выполнения суперпользователем?
5. Где хранится список пользователей и групп пользователей?
6. Как можно создать нового пользователя в системе?
7. Как можно включить пользователя в новую группу?
8. Когда вступают в силу изменения в списке групп пользователя?
9. Какие пользователи могут запускать команду `su` в ALT Linux?
10. Что такое суперпользователь системы?
11. Что такое псевдопользователи, и зачем они нужны?
12. Как посмотреть права доступа к конкретному файлу?
13. Как изменить права доступа к файлу?
14. Что такое потоки ввода/вывода? Как можно перенаправить поток ввода, поток вывода?
15. Что такое скрипт, как создать скрипт и разрешить его выполнение?
16. Что такое переменная окружения, как посмотреть значение переменной окружения?
17. Как определить и использовать переменную *shell*?
18. Какие управляющие конструкции доступны в языке командного интерпретатора?
19. Что такое регулярное выражение?
20. Какие основные конструкции используются в регулярных выражениях?
21. Что такое пакет *RPM*?
22. Какая информация хранится в заголовке пакета *RPM*?
23. Что такое репозиторий пакетов?
24. Как найти в репозитории пакет, содержащий нужную программу?
25. Как запустить, остановить, перезапустить сервис?
26. Какие сервисы настроены на автоматическое выполнение при загрузке системы?
27. Как включить и исключить сервис из списка автоматического выполнения?
28. Как организовать периодическое выполнение программ?
29. Объясните порядок работы скриптов, использованных в лабораторной работе для получения и вывода данных.
30. Поясните, под какими учётными записями пользователей выполняются установленные в лабораторной работе скрипты и как ограничен доступ к используемым ими каталогам выбранными правами доступа.

Литература

1. Георгий Курячий, Кирилл Маслинский
   «Введение в ОС Linux» - учебное пособие по работе с операционной системой Linux, распространяется на условиях лицензии GNU FDL:
   http://heap.altlinux.org/issues/textbooks/LinuxIntro.george/index.html

2. ALT Linux снаружи. ALT Linux изнутри. Под ред. Кирилла Маслинского, М.: ALT Linux; Издательский дом ДМК-пресс, 2006 г. - 416 стр.
   Доступна на условиях лицензии GNU FDL,
   http://heap.altlinux.org/alt-docs/compactbook/index.html

3. Робачевский А.М., Немнюгин С.А., Стесик О.Л. Операционная система UNIX. – 2 изд., СПб.: BHV – Санкт-Петербург, 2005. – 636 с.

4. Забродин Л.Д. UNIX. Введение в командный интерфейс. – М.: ДИАЛОГ-МИФИ, 1994. – 144 с.

5. Керниган Б.В., Пайк Р. UNIX – универсальная среда программирования: Пер. с англ. – М.: Финансы и статистика, 1992. – 304 с.

6. Дансмур М., Дейвис Г. Операционная система UNIX и программирование на языке Си: Пер. с англ. – М.: Радио и связь, 1989. – 192 с.

7. Торвальдс Л., Даймонд Д. Ради удовольствия: рассказ нечаянного революционера. — М.: ЭКСМО-Пресс, 2002. — 288 с.
   http://www.lib.ru/LINUXGUIDE/torvalds_jast_for_fun.txt

8. Advanced Bash-Scripting Guide, перевод на русский язык
   http://www.opennet.ru/docs/RUS/bash_scripting_guide/

9. Advanced Bash-Scripting Guide
   http://tldp.org/LDP/abs/html/