Лабораторная работа № 1

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

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.

Структура файловой системы.

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

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

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

К основным поддерживаемым в Linux файловым системам относятся:

- *Ext2* — файловая система, изначально разработанная для систем Linux. Сравнительно простая в реализации. Сейчас используется в основном во встраиваемых системах, например, в маршрутизаторах, сотовых телефонах, в качестве корневой файловой системы сетевых накопителей бытового уровня и т.п.
- *Ext3* — дальнейшее развитие *Ext2*, файловая система с поддержкой журналирования. Совместима с *Ext2*. При хранении большого числа файлов в каталогах использование *Ext3* неэффективно.
- *Ext4* — дальнейшее развитие *Ext3*, файловая система с поддержкой журналирования. Совместима с *Ext2* и *Ext3*. Более эффективна при работе с большим числом файлов в каталогах.
- *XFS* — журналируемая файловая система, разработанная для рабочих станций Silicon Graphics (SGI) с операционной системой IRIX. Изначально спроектирована для работы с мультимедийными файлами большого размера, эффективна при использовании расширенных списков контроля доступа к файлам. При использовании *XFS* крайне желательна надёжно работающая аппаратная платформа и наличие резервирования электропитания оборудования.
- *Btrfs — журналируемая файловая система, разрабатываемая как замена файловых систем Ext3/Ext4. Обеспечивает эффективную работу с файлами небольшого размера, каталогами с большим числом файлов, имеет возможность прозрачного сжатия хранящихся данных. Поддерживает создание снимков состояния файловой системы, возможность размещения файловой системы на нескольких физических устройствах, оптимизирована под работу с твердотельными дисками*

(SSD). В настоящее время Btrfs можно считать достаточно стабильной для использования в настольных системах; для серверных систем пока её использование не рекомендуется.

- *JFS* (*Journaled File System*) — журналируемая файловая система, изначально разработанная корпорацией IBM для ОС AIX.
- *ReiserFS* — журналируемая система, разработанная Гансом Рейзером (Hans Reiser). Оптимизирована для работы с каталогами, содержащими большое количество файлов, а также для хранения небольших по размеру файлов.
- *ISOFS* (*iso9660*) — файловая система, разработанная для дисков CD, но достаточно часто встречающаяся и на дисках DVD. Имеет ограничение максимального размера файла в 2 Gb.
- *UDF* — файловая система, обычно используемая для дисков DVD.
- *VFAT* — развитие файловой системы MS DOS с добавленной поддержкой длинных имён файлов. Из достоинств файловой системы — простота её реализации. Используется в основном на съёмных носителях данных типа USB Flash. Для разделов дисков, больших 32 Gb, использование *VFAT* крайне неэффективно.
- *NTFS* — файловая система, используемая в ОС Microsoft Windows NT и более поздних. В отличие от *VFAT*, использует журналирование и имеет систему контроля прав доступа к файлам. Использование данной файловой системы в Linux ограничено из-за отсутствия открытой документации по архитектуре файловой системы и сильной зависимости её реализации от архитектуры ОС MS Windows.

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

- *procfs* — файловая система, позволяющая обращаться к ряду структур 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.
- GlusterFS — сетевая кластерная файловая система, позволяющая объединить дисковые накопители нескольких систем в доступное по сети единое общее хранилище, с обеспечением распределения файлов по входящим в кластерную файловую систему серверам и дублированием файлов на дисках нескольких разных систем.

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:

- `.txt` — для текстовых файлов;
- `.conf, cnf` — для файлов настроек программ, обычно представляющих собой текстовые файлы специального формата;
- `.log` — для журналов работы прогамм, обычно также являющихся тестовыми файлами;
- `.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. В ряде случаев файлы могут иметь несколько последовательных расширений – например, сжатый утилитой `gzip` файл в формате HTML может иметь расширение `.html.gz`. Однако для самой операционной системы расширения файлов никакой смысловой нагрузки не несут, никаких ограничений ни на размер расширений, ни на их количество 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 файловых системах *nix информация о деталях размещения данных обычного файла на носителе информации хранится во внутренних таблицах файловых систем, запись о конкретном файле носит название *inode* (от

informational node, информационный узел), или индексный дескриптор. Каждый *inode* в рамках файловой системы имеет свой уникальный номер.

Помимо информации о размещении данных файла на физическом носителе, в *inode* хранится также его тип (файл, каталог, пр.), время создания, изменения и последнего доступа к файлу, права доступа к нему, и дополнительная информация в зависимости от файловой системы.

Символьное имя файла связывается с определённым *inode* по его номеру, эти записи хранятся в таблицах – каталогах. Каталог с точки зрения системы является файлом специального формата, для хранения данных каталога (таблицы соответствия символьных имён файлов числовым значениям *inode*) используется *inode* в файловой системы. В рамках структуры *inode* записи для обычных файлов и записи о каталогах отличаются хранящимся в *inode* типом файла.

Одному *inode* может соответствовать несколько разных символьных имён файлов в разных записях как одного, так и разных каталогов. В *inode* есть счётчик текущего числа символьных имён, которые ссылаются на данный *inode*. При добавлении символьного имени к *inode* счётчик увеличивается, при удалении символьного имени из каталога – уменьшается. Если счётчик оказывается равным нулю — т. е. в файловой системе нет больше символьных имён, ссылающихся на данный *inode*, и никакой процесс не использует данный файл, то файл считается удалённым и *inode* (выделенное для размещения данных файла место на носителе информации) очищается.

Такие добавочные имена для одного файла называются *жёсткими ссылками* (hard link). Все жёсткие ссылки равноправны, доступ к данным файла (открытие на чтение или запись) возможен по имени любой ссылки.

Т.к. жёсткие ссылки ссылаются на файл по номеру его *inode*, существовать они могут только в пределах одной файловой системы. Создать в каталоге жёсткую ссылку на файл, находящийся на другой по отношению к каталогу файловой системе, невозможно. Также нельзя создавать жёсткие ссылки на каталоги – это бы позволило делать закольцованные структуры каталогов внутри файловой системы.

Символьная ссылка (или символическая ссылка, soft link) представляет собой имя в каталоге, ссылающееся не на *inode* файловой системы, а на другой файл по его символьному имени. В отличие от жёстких ссылок символьная ссылка может ссылаться на файл или каталог, на данной или на другой файловой системе, и также может ссылаться на несуществующий файл (т.н. "битая" ссылка).

С точки зрения системы символьная ссылка – это также файл специального формата; для хранения данных символьной ссылки (т. е. пути к файлу, на который она ссылается) также используется *inode* файловой системы. Обращение к данным файла (открытие на чтение или запись) по

имени символьной ссылки не отличается от обращения по обычному имени, но дополнительно есть возможность изменения самой символьной ссылки – указание для неё другого имени, на которое она ссылается.

Таким образом, каталоги и символьные ссылки являются особыми типами файлов. Помимо обычных файлов, каталогов и символьных ссылок в *nix-системах есть ещё 4 специальных типов файлов. (Отметим, что жёсткая ссылка особым типом файла не является, это просто запись в каталоге об обычном существующем в файловой системе файле.) Этим специальным типам файлов не требуется хранить какую-либо информацию на носителе информации – выделяемые для них *inode* используются только для размещения информации о типе файла, времени создания, изменения и последнего доступа к файлу, прав доступа к нему, и т.п.

Файл устройства представляют собой запись в каталоге, ссылающуюся на интерфейс какого-либо устройства компьютера в структурах ядра (например, клавиатуры, дискового накопителя, аудио-устройства, и т.п.). В зависимости от характера устройства файлы устройства бывают символьные (позволяющие читать и/или записывать по одному байту) или блочные (позволяющие читать или записывать только сразу блоки данных определённого размера).

Файлы устройств обычно создаются при загрузке системы для в соответствии с имеющимися аппаратными средствами и далее создаются и удаляются при подключении и отключении внешних устройств отслеживающими такие события системными программами. Самостоятельно создавать их, как правило, не требуется.

С файлами устройств можно работать также как и с обычными файлами — т. е. считывать из них и записывать в них данные стандартными библиотечными функциями работы с файлами, использовать имена файлов устройств в стандартных программам и утилитах, и т.п.

Три остальных специальных типа устройств используются для организации межпроцесного взаимодействия и встречаются существенно реже. Это *именованные каналы ввода-вывода* (named pipe), *сокеты ввода-вывода* (socket) и *двери* (door). Файлы данных типов обычно создаются при запуске использующих их для обмена данными программ, обычно в каталогах временных файлов `/run`, `/var/run`, `/tmp` . Самостоятельно ни создавать, ни удалять их обычно не нужно.

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.

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

В общее дерево каталогов могут быть смонтированы файловые системы всех поддерживаемых в системе типов (что определяется наличием в системе драйверов (модулей) соответствующих файловых систем). Эти системы могут иметь разную архитектуру, разные наборы хранящихся в их *inode* параметров, или вообще не использовать механизм *inode*, как файловые системы FAT/VFAT, или сетевая файловая система CIFS. Унифицированный доступ к всему дереву каталогов в рамках операционной системы предоставляется через отдельную подсистему ядра – виртуальную файловую систему *VFS* (Virtual File System, или Virtual Filesystem Switch). В рамках *VFS* реализуются стандартные библиотечные функции по работе с файловой системой (т. е. такие функции языка C, как *open()*, *read()*, *write()*, *mkdir()*, и т. д.). Запросы прикладных программ к этим функциям внутри VFS транслируются к драйверам файловых систем, которые в случае отсутствия в соответствующих файловых системах тех или иных функций (например, прав доступа к файлам в файловой системе VFAT) их эмулируют – скрывая различия между файловыми системами от пользовательских программ.

При монтировании файловых систем можно указывать дополнительные параметры, влияющие на поведение смонтированной файловой системы. В частности, файловые системы могут быть смонтированы в режиме «только для чтения» – в этом случае изменения на них файлов и каталогов будут невозможны, при попытке записи соответствующие функции *VFS* вернут ошибки доступа. Такой режим монтирования широко используется в опе-

рационных системах мобильных устройств, где монтирование в режиме «только для чтения» файловых систем с системными файлами обеспечивает дополнительную защиту от нежелательной модификации служебных данных пользователями. При необходимости обновления системных файлов в этом случае программа-установщик изменяет параметры монтирования файловых систем, делая их доступными для записи, вносит в системные файлы нужные изменения, и возвращает файловые системы обратно в режим доступа только на чтение.

Каталог, в который монтируется файловая система, не обязан быть пустым и может содержать файлы и подкаталоги – в этом случае смонтированная в него файловая система обычно перекрывает содержимое каталога нижележащей файловой системы и делает обращения к имеющимся в нём файлам невозможным. Однако определённые типы файловых систем (объединяющие (union), или оверлейные (overlay) файловые системы) дают возможность объединить содержимое нижележащей файловой системы и смонтированной поверх неё другой файловой системы (или последовательно нескольких файловых систем). В этом случае содержимое каталогов нижележащих файловых систем остаётся доступным, но изменения записываются в последнюю смонтированную файловую систему. При удалении файлов это также учитывается в последней смонтированной файловой системе; оригинальные файлы в нижележащих файловых системах не изменяются, но перестают отображаться и становятся недоступными.

Такое объединение файловых систем позволяет, например, использовать в качестве нижележащей файловую систему, доступную только для чтения, и обеспечить запись изменений в смонтированную поверх неё оверлейную файловую систему. Последняя может быть временной и располагаться в виртуальной памяти – изменения при этом будут доступны до перезапуска системы, после перезагрузки система автоматически вернётся в изначальное состояние.

Также есть возможность смонтировать в каталог файловой системы как корневой каталог, так и произвольный подкаталог её же самой – как в режиме «только для чтения», так и с возможностью записи, как с учётом смонтированных в подкаталоги файловой системы других файловых систем, так и без этого. Вместе с оверлейными файловыми системами такие возможности позволяют эффективно и быстро создавать файловые системы для использования в виртуальных окружениях и контейнерах.

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

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

- `/bin/` — каталог с основными программами и утилитами. Расположенные в этом каталоге программы жизненно необходимы для функционирования операционной системы и её нормальной загрузки;
- `/boot/` — каталог с файлами ядра операционной системы и начального загрузчика, файлами конфигурации загрузчика операционной системы. В подкаталог /boot/efi/ для современных компьютеров обычно монтируется файловая система начального загрузчика UEFI;
- `/dev/` — каталог с файлами устройств. В старых *nix-системах в каталоге `/dev/` размещались файлы устройств для практически всего поддерживаемого системами оборудования, и размер этого каталога был достаточно большим. В современных дистрибутивах Linux в каталог `/dev/` монтируется файловая система *udevfs*, и в нём присутствуют только файлы реально подключенных и используемых в системе устройств;
- `/etc/` — каталог с файлами конфигурации системы и программ. Практически все настройки системы хранятся в текстовых файлах, которые можно легко просмотреть и изменить обычным текстовым редактором. Как правило, помимо самих настроек в файлах конфигурации размещаются комментарии и описания этих настроек. Обычно комментарии начинаются с символов `#` (октоторп) или `;` (точка с запятой);
  - `/etc/X11/` — каталог с конфигурацией *X Window System, version 11*;

  - /etc/opt/ — каталог с конфигурацией программ из /opt/;
- /home/ — каталог для домашних каталогов пользователей. Каждому пользователю системы выделяется т.н. домашний каталог — каталог, в котором хранятся личные файлы пользователя, персональные настройки программ и т.п. Например, если в системе есть пользователь student, то его домашний каталог будет находится в /home/student/ ;
- /lib/ — основные библиотеки системы — т.е. библиотеки, использующиеся программами из каталогов /bin/ и /sbin/. В каталоге /lib/modules/ размещаются загружаемые модули ядра операционной системы;
- /lib64/ — каталог для системных библиотек, с указанием архитектуры системы. В данном случае — это каталог для 64-битных библиотек 64-разрядных процессоров с архитектурой Intel/ARM. Такие каталоги не обязательны и в ряде систем могут отсутствовать;
- /lost+found/ — каталоги с таким названием могут присутствовать в корне разделов файловых систем. В ходе проверок файловых систем на целостность после сбоев (например, после отключения питания работающей системы), в эти каталоги помещаются обнаруженные «потерянные» файлы. В нормально работающих системах должны быть пустыми; для некоторых файловых систем (*Ext2*, *Ext3*, *Ext4*) автоматически создаются 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. В настоящее время в настольных системах каталог /media/ обычно не используется, монтирование съёмных носителей выполняется с правами текущего пользователя системы в

подкаталоги внутри каталога /var/run/media/<имя пользователя>/ .

- /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/, обычно 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/local/` — структура каталогов третьего уровня для программ, устанавливаемыми из исходных текстов. Содержит подкаталоги с данными конкретных программ;
  - `/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/opt/` — структура каталогов третьего уровня для программных пакетов, устанавливаемых в каталог `/opt/` ;
  - `/var/spool/` — каталог с данными, которые ожидают обработки, например:
    - `/var/spool/mail/` — входящие почтовые ящики пользователей;
    - `/var/spool/cups/` — очереди документов для печати системы *CUPS* (*Common Unix Printing System*);
  - `/var/run/` — информация о запущенных и работающих неинтерактивных программах;

- `/var/tmp/` — различные временные файлы. В отличие от `/tmp/`, содержимое этого каталога должно сохраняться между перезагрузками системы.

Пользователи и процессы в системе.

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

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

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

пользователя, а дальше и этот процесс, и все создаваемые им дочерние процессы изменить свои *UID* не могут. Имеется аналогичный системный вызов и для смены *GID*.

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

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

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

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

Кроме суперпользователя и обычных пользователей в системе существует набор т.н. псевдопользователей — непривилегированных пользователей, с правами которых работают различные системные программы. Как правило, псевдопользователям не назначен командный интерпретатор, а их домашний каталог — это тот каталог, в который соответствующие программы могут писать свои данные. Если при использовании учётной записи псевдопользователя запуск от этой учётной записи командного интерпретатора не предполагается, в файле `/etc/passwd` вместо командного интерпретатора указывается пустое устройство `/dev/null`; если не требуется хранение настроек в домашнем каталоге — то вместо него обычно указывается стандартный пустой каталог `/var/empty/`.

Системные программы обычно запускаются с привилегиями суперпользователя и после инициализации изменяют с помощью системного вызова `chuser()` свой идентификатор пользователя на непривилегированный.

В качестве дополнительной меры безопасности существует возможность изменения в настройках запущенных процессов указателя на корневой ка-

талог. 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, про-

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

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

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

$ ls -l /var
итого 72
drwxr-xr-x  2 root root   4096 Апр 19  2007 adm
drwxr-xr-x  4 root root   4096 Фев 15 08:32 cache
drwxr-xr-x  2 root root   4096 Апр 19  2007 db
dr-xr-xr-x  2 root root   4096 Апр 19  2007 empty
drwxr-xr-x 11 root root   4096 Фев  9 15:29 lib
drwxr-xr-x  2 root root   4096 Апр 19  2007 local
drwxr-xr-x  6 root root   4096 Фев 20 07:32 lock
drwxr-xr-x 14 root root   4096 Фев 20 07:32 log
lrwxrwxrwx  1 root root     10 Фев  5 13:22 mail -> spool/mail
drwxr-xr-x  2 root root   4096 Апр 19  2007 nis
drwxr-x---  2 root nobody 4096 Апр 19  2007 nobody
drwxr-xr-x  2 root root   4096 Апр 19  2007 opt
drwxr-xr-x  2 root root   4096 Апр 19  2007 preserve
drwxr-xr-x  5 root root   4096 Апр 19  2007 resolv
drwxr-xr-x  5 root root   4096 Фев 17 03:38 run
drwxr-xr-x  6 root root   4096 Фев 20 07:32 spool
drwxrwxrwt  2 root root   4096 Апр 19  2007 tmp
drwxr-xr-x  3 root root   4096 Фев 15 09:24 www
drwx------  2 root root   4096 Апр 19  2007 yp
$ ls -l /bin/su
-rws--x--- 1 root wheel 23712 Окт 18  2006 /bin/su
$ ls -l /usr/bin/crontab
-rwx--s--- 1 root crontab 39424 июн 30  2022 /usr/bin/crontab
$ ls -ld /var/spool/cron/
drwx-ws--T 2 root crontab 4096 июн 30  2022 /var/spool/cron/
```

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

документации к команде `ls`.

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

Первый символ из этого ряда обозначает тип файла. Символ `-` (дефис) означает, что это — обычный файл, который не является каталогом (в этом случае первым символом было бы `d`), символьной ссылкой (было бы `l`) или псевдофайлом устройства (было бы `c` или `b`). Следующие три символа (`rw-`) представляют собой права доступа, предоставленные владельцу `student`. Символ `r` — сокращение от read (*англ.* читать), а `w` — сокращение от write (*англ.* писать). Таким образом, `student` имеет право на чтение и запись (изменение) файла `file.txt`.

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

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

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

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

Для `/var/adm` первое поле содержит `drwxr-xr-x`. Это каталог, владелец которого — `root` — имеет права `rwx` — т.е. может читать, писать и переходить в этот каталог. Пользователи из группы `root` имеют права `r-x` — т.е.

могут читать содержимое каталога и переходить в него. Те же права и у всех остальных пользователей.

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

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*. Флаг *sticky bit* ограничивает операции с файлом в каталоге (его переименование и удаление) пользователем, являющимся владельцем файла. Так как каталог `/var/tmp/` предназначен для временных файлов, тов него разрешена запись всем пользователям. При создании в нём каким-либо пользователем файла владельцем файла станет этот пользователь. И, из-за установленного на каталог *sticky bit* далее переименовать или удалить этот файл может только этот пользователей – хотя права на запись в каталог есть у всех.

Необходимость задания флага *sticky bit* встречается достаточно редко, и в выводе `ls -l` отдельного поля для него нет. В случае, если флаг задан — в выводе прав доступа меняется последний символ из трёх символов прав для всех других пользователей. При этом, если право на выполнение («`x`») установлено, то символ «`x`» меняется на «`t`», если право на выполнение не установлено, то символ «-» меняется на «`T`».

Флаг *sticky bit* имеет значение только для каталога; установить его для файла возможно (т. к. в структуре *inode* место под него есть), но никакого действия не несёт.

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 означает наличие у файла флага *SUID* (*Set-User-ID*) *bit*, при этом 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.

Флаг *SUID bit* требуется применять крайне осмотрительно, ошибочное задание флага на выполняемый файл может привести к нежелательному повышению привилегий в системе. Флаг имеет смысл только для исполняемых файлов со скомпилированными в двоичный код программами, для остальных файлов он бесполезен и использоваться не будет. Задание флага *SUID bit* для каталогов возможно, но никакого действия не несёт.

Также существует третий флаг – *SGID* (*Set Group-ID*) *bit*. В выводе команды ls -l его наличие показывается в составе прав для группы владельца, заменой права на выполнение «x» на букву «s», или, в случае отсутствие права на выполнение для группы — на «S».

Для исполняемых файлов смысл задания данного флага аналогичен заданию *SUID bit*, но при запуске программы она будет выполняться не с правами текущей *группы пользователя*, а с правами *группы пользователей файла*. В качестве примера можно привести программу /usr/bin/crontab, предназначенную для редактирования списка фоновых задач пользователей. Её выполнение разрешено только пользователям группы crontab, и при её запуске она будет выполняться с правами пользователя данной группы — а не текущей группы пользователей.

Флаг *SGID bit* также может устанавливаться на каталоги, в этом случае создаваемые пользователями в таком каталоге файлы будут принадлежать не текущей группе пользователя, а группе пользователей каталога. Например, для хранения создаваемых пользователями списков фоновых задач предназначен каталог /var/spool/cron. Этому каталогу назначена группа пользователей crontab, и для него установлен флаг *SGID bit* – создаваемые в этом каталоге файлы расписаний всегда будут назначаться группе пользователей crontab. Кроме того, при создании подкаталогов внутри ка-

талога с установленным флагом *SGID bit* он будет автоматически добавляться и для этих новых подкаталогов.

Комбинация флагов *Sticky bit* и *SGID bit* может использоваться вместе для задания иерархии каталогов, для которых есть права на запись у группы пользователей, при этом входящие в группу пользователи не могут изменять или удалять файлы, созданные другими пользователями. Такая комбинация часто используется для каталогов журналов работы программ в `/var/log/` .

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------`.

Для жёстких ссылок отдельных прав доступа нет – т. к. жёсткая ссылка является просто записью в каталоге, ссылающейся на какой-то *inode*. Права доступа к файлу, на который задана жёсткая ссылка, определяются установленными для *inode* владельцем и группой файла, и правами доступа к нему.

Для символьных ссылок существует *inode* самой символьной ссылки, и *inode* файла (или каталога), на который эта ссылка указывает. Права доступа на *inode символьной ссылку* определяют возможность изменения символьной ссылки — т. е. задания её нового (другого) значения. При обращении же к файлу или каталогу, на который указывает символьная ссылка, проверяются права в *inode* соответствующего файла или каталога. Для символьных ссылок флаги *SUID bit*, *SGID bit* и *Sticky bit* задавать можно, но какого-либо смысла они не несут.

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

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

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

Общий формат вызова команды выглядит следующим образом:

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

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

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

После имени команды, при необходимости, указываются ключи. Ключ — параметр команды, который влияет на результат её выполнения. Часто используещиеся ключи — короткие, односимвольные; для требующихся реже длинных ключей используются слова или сокращения. Короткие ключи начинаются с символа - (дефис), длинные — с двух символов - (дефис). Короткие ключи часто дублируются длинными — для повышения удобства чтения и самодокументирования скриптов. После ключей может допускаться указание дополнительных параметров, для длинных ключей такие параметры принято записывать через знак = (равно). Несколько односимвольных ключей разрешается объединять вместе: например, вместо

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

можно записать:

```
$ ls -la
```

Порядок ключей, как правило, не важен.

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

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

Команды могут использовать различные ключи и параметры. Запоминать все возможные комбинации формата вызова каждой программы

невозможно и бессмысленно. Поэтому в системе доступны описания и подсказки по использованию практически каждой утилиты и программы.

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

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

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

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

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 список команд из истории можно командой history. В качестве необязательного параметра можно задать число последних команд, которые и будут выведены на экран. В системе история команд сохраняется в файле ~/.bash_history . Количество команд, которое по-умолчанию выводится командной history , и общее количество запоминающихся последних команд зависит от настроек системы.

При работе с системой один из каталогов является текущим. В начале сеанса работы текущим каталогом становится домашний каталог пользователя, далее его можно изменять командами смены текущего каталога. Используемые в аргументах команд имена файлов и каталогов могут отсчитываться системой как от корневого каталога, так и от текущего. В первом случае говорится об абсолютных именах файлов или каталогов, во втором – об относительных. Абсолютные имена файлов начинаются с символа / (слеш), включают в себя все родительские каталоги и отсчитываются от корня файловой системы. Относительные имена не начинаются с символа / и отсчитываются от текущего каталога. В каждом каталоге существуют записи–ссылки на текущий каталог, со специальным именем . (точка), и на вышестоящий каталог, со специальным именем .. (две точки). Для корневого каталога ссылка на вышестоящий каталог через имя . соответствует самому корневому каталогу. Два и более символов / (слеш) подряд в именах файлов допустимы и сокращаются до одного. Имя каталога может включать в себя заключительный символ / (слеш), который игнорируется.

Например,

/home/student/Docs	Абсолютный путь к файлу (или каталогу) Docs в каталоге /home/student .
/home/student/Docs/	Абсолютный путь к каталогу Docs в каталоге /home/student .
file	Относительный путь к файлу (или каталогу) file в текущем каталоге.
./file	Относительный путь к файлу file , отсчитываемый от текущего каталога.
../student/file	Относительный путь к файлу file , располагающему в каталоге student на уровень выше текущего (т. е. отсчитываемый от текущего каталога через каталог вышестоящего уровня).
/bin	Абсолютный путь к каталогу /bin .
/bin/	Абсолютный путь к каталогу /bin, с указанием опционального завершающего слеша в имени каталога.
../../bin	Относительный путь к файлу или каталогу bin, размещающемуся на два уровня выше текущего.
../../bin//../home/	Относительный путь к каталогу home (подняться на два каталога от текущего каталога , войти в каталог bin, подняться уровнем выше и войти в каталог home). При разборе пути операционной системой двойной слеш после bin/ будет сокращён до одинарного.

Во всех приведённых ниже примерах команд символы $ или # означают

приглашение командной строки. Вводить их не нужно. <имя файла> — имя произвольного файла в системе, абсолютное или относительное.

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/`

Вывести имя текущего каталога 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 -ld`	Вывод в полном формате текущего *каталога*, с указанием прав доступа на него, владельца, группы, размера, даты создания и пр.
`$ ls -ld /etc`	Вывод в полном формате записи указанного в команде *каталога* (а не его содержимого).
`$ 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.
`$ ls -i`	Вывести номера *inode* для файлов в текущем каталоге.
`$ ls -il`	Добавить номера *inode* в вывод «`ls -l`».

Для копирования файлов или каталогов используется команда `cp`, аргументами которой являются имя копируемого файла и имя создаваемого нового файла — копии существующего:

`$ cp a b`	Скопировать файл `a` в файл `b` в текущем каталоге.
`$ cp a b/c`	Скопировать файл `a` в каталог `b/` под именем `c` (каталог `b` при этом должен существовать и быть доступен для записи).
`$ cp /bin/ls .`	Скопировать файл `/bin/ls` в текущий каталог с тем же именем (`ls`).
`$ cp -r b/ b1/`	Рекурсивно скопировать каталог `b/` из текущего каталога в каталог `b1/` . Если при этом каталог `b1`/ уже существует — каталог `b/` скопируется в подкаталог `b/` внутри `b1/` .
`$ cp a b c d/`	Скопировать файлы `a`, `b` и `c` в каталог `d/` .
`$ cp -a a b`	Скопировать файл или каталог `a` в файл или каталог `b` с сохранением времени создания и прав файлов (ключ `-a` включает в себя ключ `-r` для рекурсивного копирования).
`$ cp -l a b`	Создать жёсткую ссылку на файл `a` с именем `b`. Копирование содержимого файла при этом не выполняется, для *inode* файла `a` создаётся новое имя (жёсткая ссылка) `b` .

Для перемещения и переименования файлов или каталогов используется команда `mv`:

`$ mv a b`	Переименовать файл (или каталог) `a` в файл (или каталог) `b` в текущем каталоге.
`$ mv a b/c`	Переместить файл `a` в каталог `b/` под именем `c` (каталог `b/` при этом должен существовать и быть доступен для записи).
`$ mv /bin/ls .`	Переместить файл `/bin/ls` с тем же именем в текущий каталог.
`$ mv a b c d`	Переместить файлы `a`, `b` и `c` в каталог `d/` .

Команда `mv` изменяет имя в каталоге для *inode* файла (или создаёт имя для этого *inode* в другом каталоге и удаляет имя в исходном). Данные в *inode* (кроме времени доступа к *inode*) при этом не меняются. Если пути к исходному и конечному файлам для команды `mv` указаны для разных файловых систем, то выполнить переименование файла команда не может. В этом случае в зависимости от конкретной реализации команды `mv` и настроек операционной системы может или выполниться *копирование* файла с одной файловой системы на другую и далее *удаление* исходного файла, или выдача ошибки о попытке перемещения файла между разными файловыми системами.

Для создания каталога используется команда
`mkdir <имя каталога>`.
Если каталог уже существует, будет выдана ошибка.

Можно создать сразу поддерево каталогов, использую ключ `-p`:
`mkdir -p a/b/c`

В этом случае недостающие каталоги будут созданы, а ошибки при создании существующих каталогов, включая каталог `c` – проигногированы.

Для удаления пустого каталога используется команда
`rmdir <имя каталога>`.
Удалить можно только пустой каталог, если в каталоге есть файлы или каталоги (кроме встроенных ссылок `.` на текущий каталог и `..` на вышестоящий каталог) – их надо удалить до вызова `rmdir`. Также можно удалить каталог с файлами рекурсивно через команду `rm`.

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 (и, в любом случае, сама возможность такого восстановления зависит от используемых файловых систем).

`$ rm file.txt`	Удалить файл `file.txt` в текущем каталоге.
`$ rm *.txt`	Удалить все файлы, заканчивающиеся на `.txt`, в текущем каталоге.
`$ rm -f *`	Удалить все файлы в текущем каталоге, не запрашивая разрешений.
`$ rm -r directory/`	Рекурсивно удалить все файлы в каталоге `directory/` и сам каталог `directory/`.

`$ rm -rf *`	Рекурсивно удалить все файлы и каталоги из текущего каталога, не запрашивая подтверждения. Данная команда, отданная от имени суперпользователя и в корневом каталоге, удалит всю файловую систему **без дополнительных вопросов и возможности отмены действия** – поэтому в ряде дистрибутивов в реализации команды `rm` специально внесены изменения, запрещающие такое поведение).

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

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

`$ cat /etc/passwd`	Вывести на экран содержимое файла `/etc/passwd`.
`$ more /etc/passwd`	Вывести на экран содержимое файла `/etc/passwd`. Если вывод не будет помещаться на одном экране — вывести начало файла и ждать нажатия любой клавиши для следующей страницы.
`$ less /etc/passwd`	Вывести на экран содержимое файла `/etc/passwd`. Если вывод не будет помещаться на одном экране — вывести начало файла и позволить пользователю просмотреть его, используя прокрутку клавишами управления курсором. Для завершения работы команды `less` следует нажать клавишу `<q>`.
`$ head /etc/passwd`	Вывести первые 10 строк файла `/etc/passwd`.
`$ head -5 /etc/passwd`	Вывести первые 5 строк файла `/etc/passwd`.
`$ tail /etc/passwd`	Вывести последние 10 строк файла `/etc/passwd`.
`$ tail -1 /etc/passwd`	Вывести последнюю строку файла `/etc/passwd`.

Правами доступа к файлам можно управлять командой `chmod`:

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

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

В настоящее время пароль не должен содержать менее 8 символов. Эти символы не должны быть одинаковыми, пароль не должен содержать только цифры или быть словарным словом. Все подобные, слабые пароли легко определяются современными программами для взлома систем в автоматическом 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`.
`# su -l user`	Запустить командный интерпретатор с правами пользователя `user`.
`# useradd user`	Добавить учётную запись пользователя `user`. При этом создаются необходимые записи в файлах `/etc/passwd` и `/etc/group`, а также домашний каталог пользователя. Пароль новому пользователю не назначается, и войти в систему до его задания он не может.
`# userdel user`	Удалить учётную запись пользователя `user`. Файлы, принадлежащие пользователю, при этом не удаляются.
`# passwd user`	Задать пароль пользователя `user`. Суперпользователю знать старый пароль `user` для его смены не нужно.
`# chmod <права> file`	Изменить права на файл. `root` может изменить права доступа к любому файлу.
`# chown user file`	Изменить владельца файла на пользователя `user`.
`# chown :group file`	Изменить группу файла на `group`.
`# chown user:group *`	Изменить владельца и группу всех файлов в текущем каталоге.
`# chown –R user:group *`	Рекурсивно изменить владельца и группу всех файлов в текущем каталоге и подкаталогах.
`# shutdown`	Выключить систему. Без дополнительных ключей команда `shutdown` останавливает систему, не отключая её питание. Удалённо выключить систему можно, но включить её после этого возможно только внешними средствами.
`# halt`	Выключить систему, не отключая её питание (аналогично вызову `shutdown`).

`# shutdown -h`	Выключить систему, и отключить её питание.
`# poweroff`	Выключить систему, в т.ч. отключить её питание (аналогично вызову `shutdown -h`).
`# shutdown -r`	Перезагрузить систему.
`# reboot`	Перезагрузить систему (аналогично вызову `shutdown -r`).

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

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, кото-

рые 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, в отечественном проекте Astra Linux.

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 ROSA Linux, РЕД ОС 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 файлов пакетов `.i586.rpm`), 64-битных процессоров Intel и AMD (с расширением файлов пакетов `.x86_64.rpm`), 32-битной архитектуры ARMv7 (с расширением файлов пакетов `.armh.rpm`), 64-битной архитектуры ARMv8 (с расширением файлов пакетов `.aarch64.rpm`) и 64-битной архитектуры POWER8 (с расширением файлов пакетов `.ppc64le.rpm`), а также дополнительно – архитектур RISC-V, MIPS, Эльбрус v3 и Эльбрус v4. Кроме того, существуют программы, являющиеся архитектурно-независимыми — например, написанные на интерпретируемых языках. Для того, чтобы избежать дублирования пакетов с такими программами для каждой из архитектур, они упаковываются в пакеты с архитектурой *noarch*. Таким образом, для систем с 32-битными процессорами с системой команд x86 нужно использовать пакеты `.i586.rpm` и `.noarch.rpm`, для 64-битных систем Intel/AMD — `.x86_64.rpm` и `.noarch.rpm`.

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

Используя команду `rpm`, можно получать информацию о пакетах,

устанавливать, обновлять и удалять их, а также собирать пакеты с исходным кодом и компилировать их в бинарные пакеты. Для получения информации о пакетах предназначается ключ `-q`.

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

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

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

Более подробную информацию можно получить, добавив ключ `-i`:

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

Как видно, в пакете указывается, помимо его версии и релиза, также время компиляции пакета, время его установки в системе, лицензия, URL сайта разработчика программы, краткое и полное описание программы в пакете.

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

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

Для каждого файла в системе, установленного из пакета, в кэше `rpm`

хранится соответствующая запись. Всегда можно посмотреть, какому пакету принадлежит тот или иной файл или каталог. Для этого используется команда `rpm -qf`:

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

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

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

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

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

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 [p10] ftp://ftp.altlinux.org/pub/distributions/ALTLinux/p10/branch x86_64
classic
 rpm [p10] ftp://ftp.altlinux.org/pub/distributions/ALTLinux/p10/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/p10/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 март 2024 г. приведённый выше репозиторий занимал порядка 467 GiB, из них около 99 GiB занимали пакеты с исходными кодами, 71 GiB — пакеты для архитектуры ARMv8, 49 GiB — для ARMv7, 60 GiB — для i586, 81 GiB — для x86_64, 36 GiB – архитектурно-независимые пакеты. Кроме того, пакеты в репозитории постоянно обновляются (где-то 10-15 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.
```

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

ответ (y), apt-get получила новые версии пакетов и установила их в системе.

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

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

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

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

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

В данном случае была запрошена установка пакета rrd-perl. Этот пакет зависит от библиотеки librrd, которая, в свою очередь, использует библиотеки libpng12 и др. Всего установка пакета потребовала бы установку дополнительно ещё четырёх — что и предложила сделать

`apt-get`. Получив отрицательный ответ на вопрос о продолжении операции, `apt-get` отменила её.

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

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

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

В ряде случаев удаляемый пакет критически необходим для системы:

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

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

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.

Традиционно используемой 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.

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

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

В системе инициализации sysvinit сервисы запускаются скриптами, расположенными в каталоге`/etc/rc.d/init.d/` или `/etc/init.d/`. При размещении скриптов инициализации в каталоге `/etc/rc.d/init.d/` на этот каталог обычно создаётся символьная ссылка `/etc/init.d/`, что делает использование путей `/etc/init.d/` и `/etc/rc.d/init.d/` равнозначным. Для управления тем, какой скрипт и на каком уровне запускается, используется команда `chkconfig`.

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

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

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

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

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

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

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

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

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

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

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

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

Например, для сервиса lighttpd:

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

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

Изначально сервис не был запущен. По команде `service lighttpd start` он был запущен, что подтвердил последующий вывод команды `service lighttpd status`. Также успешно прошёл перезапуск сервиса (в

процессе которого он остановился и заново запустился, перечитав свою конфигурацию), и его остановка. Последняя команда `restart` не нашла работающего сервиса `lighttpd`, о чём сообщила, и потом его успешно запустила.

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

Как правило, в пакетах из состава дистрибутива присутствуют как файлы конфигурации для systemd, так и скрипты для sysvinit — при этом для управления демонами выбирается вариант, соответствующий используемой системе инициализации. В то же время для ряда программ (обычно новых) в пакетах только файлы конфигурации для systemd, а для части других (обычно старых) в пакетах есть только скрипты для sysvinit . Для работы с В последнем случае 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 P10 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` .

Протокол SSH и его возможности по работе с удалёнными системами.

Одним из широко используемых сетевых протоколов для работы с удалёнными системами является протокол SSH (от Secure Shell – «безопасная оболочка»). Данный протокол позволяет выполнять удалённое подключение к системам через сети TCP/IP, получать терминальный доступ к ним, удалённо запускать команды, передавать файлы.

Кроме того, с помощью SSH можно организовывать проксирование соединений других протоколов, пропуская их через канал SSH и позволяя перенаправлять соединения с клиентской машины через удалённый сервер.

SSH – клиент-серверный протокол, на удалённой системе запускается сервер SSH, к которому через сеть подключаются клиенты. Есть ряд реализаций протокола, для разных операционных систем. В настольных и серверных Linux-системах обычно используется реализация протокола SSH OpenSSH, для систем с ограниченными ресурсами есть более простая и менее функциональная реализация протокола Dropbear. Сборки OpenSSH (как клиента, так и сервера) доступны для Microsoft Windows, свои реализации есть для Android, IOS. В целом, все современные реализации совместимы друг с другом; для более старых реализаций могут быть проблемы

совместимости из-за отсутствия поддержки современных криптографических протоколов и отключения устаревших и небезопасных протоколов в более новых реализациях.

Протокол SSH – безопасный в том смысле, что всё передаваемые между клиентом и сервером данные защищаются криптографическими протоколами, в открытом виде через сети ничего не передаётся. Аутентификация при подключении клиента к серверу выполняется для конкретного пользователя сервера, и может проводиться как с использованием пароля пользователя, так и с использованием протоколов открытых ключей. Часть реализаций SSH позволяет также использовать другие методы аутентификации, типа Kerberos и пр.

Для аутентификации с использованием открытых ключей на стороне клиента создаётся пара из закрытого и открытого ключа, по одному из протоколов. Открытый ключ передаётся на сервер и указывается в настройках соответствующего пользователя, закрытый ключ хранится только на клиенте. Возможно дополнительная защита закрытого ключа паролем – тогда при подключении к серверу клиент SSH запрашивает пароль от закрытого ключа. Если закрытый ключ при создании пары ключей не был защищен паролем — то для аутентификации достаточно только наличия на клиенте закрытого ключа и указания для данного клиента на сервере соответствующего открытого ключа, никаких дополнительных действий от пользователя при подключении не требуется. Данный вариант широко используется для автоматизированного удалённого запуска на сервере команд с клиента.

Протокол SSH поддерживает несколько вариантов протоколов открытых ключей. Для реализации OpenSSH на настоящий момент рекомендуется использование эллиптических кривых ED25519. Также поддерживается использование ключей RSA, длиной до 4096 бит. Хотя данные ключи сейчас считаются менее надёжными, и менее удобны в использовании (т. к. их запись существенно длиннее ключей ED25519), но могут требоваться для соединения с рядом более простых и/или старых реализаций протокола, которые не поддерживают ключи ED25519. Также есть поддержка ключей DSA — но на настоящий момент их использование нежелательно из-за наличия известных уязвимостей. К сожалению, часть старых реализаций протокола SSH, особенно в старых аппаратных решениях, поддерживают только ключи DSA.

Выполнение настоящей работы предполагается на выделенном виртуальном сервере, соединение с которым устанавливается с произвольного рабочего места по протоколу SSH.

Для терминального доступа, запуска команд, перенаправления соединений в *nix-системах в целом, и в реализации OpenSSH в частности, используется команда ssh. Общий формат вызова команды –

```
ssh [-keys] <user>@<server> [command]
```

Данная команда выполняет подключение к серверу <server> (заданному или символьным именем, или адресом IP) от имени пользователя <user> на сервере <server> . Если указана команда <command>, то она выполняется на сервере после соединения с ним, если нет – выполняется подключение к серверу и запуск на нём терминальной сессии.

Управлять поведением команды ssh можно ключами. Наиболее часто используемые из них:

-v, -vv, -vvv – включение отладочного вывода. Больше „v" — более подробный вывод. Данные ключи полезны в случае возникновения проблем с работой ssh, невозможностью подключения к серверу, ошибок аутентификации.

-p <port> – подключаться к серверу <server> на порт <port>. В рамках протоколов TCP/IP каждый запускаемый на каком-либо сервере сервис принимает соединения на определённый порт TCP или UDP. Стандартный порт для протокола SSH — TCP/22, если сервер SSH принимает соединения на нестандартном порту, этот порт требуется указывать отдельно. Подробнее сети TCP/IP рассматриваются в лабораторной работе № 3.

-i /path/to/key – подключаться к серверу <server> с аутентификацией по открытому ключу, использовать закрытый ключ в указанном файле. По-умолчанию OpenSSH в *nix-системах хранит настройки пользователя в каталоге ~/.ssh (подкаталог .ssh в домашнем каталоге). Файлы ключей защищаются от несанкционированного доступа правами на каталог .ssh и на файлы закрытых ключей, при неправильных правах доступа клиент OpenSSH закрытый ключ использовать откажется. Кроме того, как говорилось выше, закрытые ключи могут быть защищены паролями.

По-умолчанию при подключении к серверу ssh ищет в каталоге .ssh/ закрытые ключи в файлах ~/.ssh/id_ed25519, ~/.ssh/id_rsa , ~/.ssh/id_dsa — если такие файлы есть, то сначала пробуется аутентификация на сервере с их использованием, и в случае неудачи или отсутствия ключей – предлагается парольная аутентификация.

Пары ключей для использования SSH создаются командой ssh-keygen.

Формат использования –

```
ssh-keygen [-t <type>] [-b <bit>] [-f /path/to/file]
```

Используемые ключи:

`-t <type>` – тип создаваемой пары ключей. Возможные значения – `ed25519`, `rsa`, `dsa`. По-умолчанию на текущий момент – `rsa`.

`-b <bit>` – длина создаваемого ключа RSA или DSA, по-умолчанию 2048 бит. Для ключей `ed25519` — не применимо.

`-f /path/to/file` – путь к файлу создаваемого закрытого ключа. Открытый ключ размещается в файле с именем `/path/to/file.pub`. По-умолчанию имя файла зависит от типа создаваемого ключа, это `~/.ssh/id_ed25519`, `~/.ssh/id_rsa` или `~/.ssh/id_dsa` .

Формат файлов закрытых ключей зависит от реализации клиента протокола SSH и его версии. Передача закрытых ключей между разными клиентами, как правило, не требуется.

Формат открытого ключа –

```
ssh-<type>  <KEY> <id>
```

Здесь `<type>` - тип ключа (`ed25519`, `rsa`, `dsa`), `<KEY>` - данные открытого ключа, длинная строка символов ASCII, `<id>` - произвольный идентификатор ключа. Обычно в качестве идентификатора используется строка в формате адреса электронной почты, из имени создавшего ключ пользователи и имени компьютера, на котором ключ был создан.

Пример открытого ключа:

```
 ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIOy3/gO/8p1sqPC2H2On2ShYPQvlsmfrUZNxmNFR6S08
user@my.domain.local
```

Для разрешения подключения к серверу с использованием открытого ключа на сервере в файл `~/.ssh/authorized_keys2` (или `~/.ssh/authorized_keys` – в зависимости от реализации протокола SSH) в домашнем каталоге нужного пользователя записывается строка с открытым ключом. В файле `authorized_keys2` может быть несколько строк с разными ключами, что позволяет соединяться с сервером с разных клиентов, на каждом из которых используется свой закрытый ключ.

Также через протокол SSH возможна передача файлов. Для этого используется утилита `scp`, общий формат вызова –

```
scp [-keys] /local/file   <user>@<server>:/remote/file
scp [-keys] <user>@<server>:/remote/file   /local/file
```

В первом случае выполняется копирование файла с клиента на удалённый сервер, во втором — с удалённого сервера на клиент.

Наиболее часто используемые ключи `scp`:

`-P <port>`, `--port <port>` – подключаться к серверу <server> на порт `<port>` (в отличии от `ssh`, здесь используется заглавная буква «`P`»);

`-r`, `--recursive` – рекурсивное копирование, используется для каталогов вместо отдельных файлов.

Остальные ключи совпадают с ключами `ssh`.

В рамках сервера SSH можно ограничивать подключающего клиента использованием только определённых команд, запрещать получать удалённый терминал, ограничивать доступ к файлам через `scp` или вообще отключать передачу файлов, и т. п. Всё это позволяет гибко предоставлять удалённый доступ к системам для конкретных пользователей и под конкретные задачи.

Выполнение лабораторной работы.

Лабораторная работа посвящена изучению основ работы с операционной системой семейства *nix и выполнению основных задач по её администрированию. Выполнение лабораторной работы предусматривает работу с удалённым сервером. Для доступа к серверу используется терминальная программа *PuTTY*.

В лабораторной работе требуется:

- ознакомиться с основами работы в операционной системе ALT Linux Server, изучить работу основных команд операционной системы;
- провести начальную настройку и подготовку операционной системы к использованию;
- ознакомиться с базовыми возможностями утилит SSH для выполнения команд на удалённых серверах и передачи файлов между системами.

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 подключения к удалённому серверу можно использовать любой клиент SSH; название и версия конкретной реализации клиента SSH зависит от используемой на рабочем месте операционной системы.

В случае использования систем Microsoft Windows, для подключения к серверу рекомендуется использовать терминальную программу *PuTTY*. Внутри сети МЭИ на период проведения данного курса она доступна для загрузки и запуска 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*.

В случае использования систем на базе Linux клиент SSH (OpenSSH) по-умолчанию уже должен быть установлен в операционной системе, для подключения к удалённому серверу следует открыть любую установленную на рабочем месте терминальную программу и запустить в её окне утилиту `ssh` с соответствующими параметрами.

Для систем на базе Android возможно или использование любого доступного в используемом каталоге приложений клиента SSH, например, TermBot (https://f-droid.org/ru/packages/org.sufficientlysecure.termbot/),или, что предпочтительнее, использование эмулятора терминала Termux (https://f-droid.org/ru/packages/com.termux/) и входящего в состав базовой системы в нём клиента OpenSSH.

Вне зависимости от используемого клиента SSH 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*.

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

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

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

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

Запустите командный интерпретатор с правами `root`.

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

Завершите сеанс `root`, выйдя из командного интерпретатора (`exit` или `<Ctrl>+<D>`). Снова запустите командный интерпретатор с правами `root`.

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

Проверьте список пользователей и групп в системе.

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

Задайте пароль для созданного пользователя.

Запустите вторую терминальную сессию. Для этого в меню окна *PuTTY* (доступного при нажатии на иконку приложения слева в заголовке окна)

выберите пункт *New Session*. Повторите настройки подключения и осуществите вход в систему под учётной записью созданного пользователя. Убедитесь в возможности получения им прав суперпользователя, запустив команду `su -l`.

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

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

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

Удалите домашний каталог пользователя `student`.

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

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

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

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

`rpm -q --changelog <имя пакета>`.

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

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

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

Браузер подключается к веб-серверу, устанавливая сетевое соединение по протоколу TCP/IP. Выбор нужного веб-сервера осуществляется по определённому адресу IP и порту TCP. По-умолчанию, для схемы http:// 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 адресу 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` или `systemctl start lighttpd` . Проверьте, работает ли сервер, выполнив команды `service lighttpd status`, `systemctl status lighttpd` . 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.

Проверьте, включен ли автоматический запуск `lighttpd` при загрузке системы, выполнив команду `chkconfig lighttpd` или `systemctl is-enabled lighttpd` . Если он выключен, включите его командой `chkconfig lighttpd on` или `systemctl enable lighttpd` .

Перезапустите систему командой `reboot`. Дождитесь загрузки сервера (время перезагрузки находится в пределах 2-3 минут). Войдите в систему.

Проверьте, запустился ли `lighttpd` после перезагрузки системы.

Веб-сервер `lighttpd` предоставляет удалённый доступ к документам, размещённым в определённом каталоге файловой системы – корневом каталоге веб-сервера). Этот каталог определяется параметром `server.document-root` из настроек веб-сервера. Найдите этот каталог, проверьте права доступа к нему.

Размещение (создание) и редактирование файлов документов не относится к задача администратора системы и должна выполняться с правами учётной записи обычного пользователя. Для этого обеспечьте доступ на запись к корневому каталогу веб-сервера для созданного ранее пользователя системы, добавив его в нужную группу или группы пользователей. Для применения настроек после изменения файла `/etc/group` выйдите и войдите заново в систему.

Разместите в корневом каталоге веб-сервера `lighttpd` файл с именем, соответствующем имени выделенного Вам виртуального сервера и расширением `.html` (`lab-NN.html`). Файл должен содержать цитату, полученную с тестового сервера SSH.

Для этого подключитесь к тестовому серверу SSH с использованием парольной аутентификации. Тестовый сервер SSH доступен по адресу `192.168.230.230`, имя пользователя совпадает с именем используемого для лабораторной работы виртуального сервера (`lab-NN`), пароль пользователя совпадает с паролем пользователя `student` из индивидуальных данных к лабораторной работе. После успешного подключения тестовый сервер сообщит команду для получения с него закрытого ключа SSH для выполнения дальнейших действий, и закроет соединение:

```
$ ssh lab-NN@192.168.230.230 help
```

(Запускаемые здесь и далее на тестовом сервере SSH команды `help`, `get-key`, `set-key`, `get-message` – разрабатывались для данной лабораторной работы и на других серверах или отсутствуют, или имеют другой смысл.)

Используя команду `scp`, загрузите с тестового сервера SSH файл закрытого ключа. Имя пользователя и пароль те же, что в предыдущем случае, имя файла закрытого ключа на тестовом сервере SSH было получено в предыдущем шаге.

Подключитесь к тестовому серверу SSH с использованием полученного файла закрытого ключа. Для этого:

- создайте с помощью команды `ssh-keygen` пару закрытого и открытого ключа. Тип ключа рекомендуется выбрать `ed25519`, место размещения ключа можно оставить предлагаемое по-умолчанию. Пароль на ключ задаётся по усмотрению.

- рассмотрите созданные файлы закрытого (имя по-умолчанию для ключа `ed25519` – `~/.ssh/id_ed25519`) и открытого (имя по-умолчанию для ключа `ed25519` – `~/.ssh/id_ed25519.pub`) ключей. Разместите в каталоге `~/.ssh/` полученный с тестового сервера SSH закрытый ключ и задайте для этого файла права доступа, аналогичные правам созданного с помощью `ssh-keygen` закрытого ключа.

- подключитесь к тестовому серверу SSH с аутентификацией по открытому ключу. Адрес тестового сервера SSH — `192.168.230.230`, имя пользователя — `manager` . Запустите на тестовом сервере команду `help`, соответствующая команда `ssh` –

```
$ ssh -i ~/.ssh/<key_file> manager@192.168.230.230 help
```

Здесь `~/.ssh/<key_file>` - имя ранее полученного файла закрытого ключа. Пароль для закрытого ключа тестового сервера не установлен.

В случае успешного подключения будет выдана краткая справка по доступным на тестовом сервере SSH командам. Если будет выдано приглашение к вводу пароля — аутентификация по открытому ключу не удалась. Имеет смысл повторить подключение, указав `ssh` выводить отладочную информацию (добавив к строке запуска ключи `-vvv`), и проверять права доступа на файл закрытого ключа.

Создайте пару закрытого/открытого ключа с типом `ed25519`. Возможно использовать ключи, созданные ранее. Установите открытый ключ на тестовый сервер SSH для пользователя `lab-NN`, использую команду тестового сервера `set-key`. Для этого запустите на тестовом сервере SSH команду `set-key` аналогично предыдущему пункту (аутентификация по полученному закрытому ключу, имя пользователя – `manager`), и на запрос команды введите строку созданного Вами открытого ключа. Требуется ввести только сам ключ, его тип (`ssh-ed25519`) и идентификатор (`<user>@lab-NN.edu.cbias.ru`) вводить не нужно.

Проверьте правильность установленного ключа для пользователя командой `get-key`.

Подключитесь к тестовому серверу SSH по имени пользователя `lab-NN`, используя аутентификацию через установленный ранее на тестовом сервере SSH открытый ключ, и запустите команду `get-message`:

```
$ ssh lab-NN@192.168.230.230 get-message
```

Здесь предполагается, что ключ в формате `ed25519` размещён с именем по-умолчанию (`~/.ssh/id_ed25519`), если ранее было выбрано другое имя файла — его требуется указать в параметре `-i` . Если для созданного закрытого ключа был задан пароль, при подключении он будет запрошен. В случае успешного выполнения команды будет выведена цитата – один из законов Мерфи.

Разместите полученную цитату в файле `lab-NN.html` в каталоге веб-сервера. Оформление файла HTML предлагается создать по желанию.

Проверьте, что созданный файл доступен из браузера, запросив его с рабочего места как `http://lab-NN.edu.cbias.ru/lab-NN.html` .

Задания на лабораторную работу.

1. Выполнить удалённую регистрацию в системе.
2. Изучить структуру каталогов сервера.
3. Посмотреть доступные команды в системе, вызвать справочное руководство по каким-либо из них.
4. Создать текстовый файл, используя редактор `vi`.
5. Используя команду `su`, получить привилегии суперпользователя системы.
6. Изменить пароли пользователя и суперпользователя системы.
7. Создать новую учётную запись пользователя.
8. Зарегистрироваться в системе под созданным в п. 7 пользователем, убедиться в возможности использования им команды `su`.
9. Удалить учётную запись пользователя.
10. Получить список пакетов, установленных в системе.
11. Настроить список репозиториев пакетов для системы *APT*.
12. Провести обновление системы до текущего состояния репозитория.
13. Установить веб-сервер `lighttpd`, запустить сервер. Проверить работу веб-сервера.
14. Настроить его автоматический запуск при загрузке системы.
15. Перезагрузить систему.
16. Убедиться, что веб-сервер `lighttpd` автоматически запустился после перезагрузки системы.
17. Настроить доступ по открытому ключу к тестовому серверу SSH.
18. Разместить файл с полученной от тестового сервера SSH цитатой на веб-сервере.

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. Какие права доступа есть для файлов и каталогов?
13. Как посмотреть права доступа к конкретному файлу или каталогу? Как изменить права доступа к файлу или каталогу?
14. Что такое пакет *RPM*?
15. Какая информация хранится в заголовке пакета *RPM*?
16. Что такое репозиторий пакетов?
17. Как найти в репозитории пакет, содержащий нужную программу?
18. Как запустить, остановить, перезапустить сервис?
19. Какие сервисы настроены на автоматическое выполнение при загрузке системы?
20. Как включить и исключить сервис из списка автоматического выполнения?
21. Какие возможности для работы с удалёнными системами предоставляет протокол SSH?
22. Какие преимущества есть у авторизации с использованием открытых ключей по сравнению с парольной авторизацией?

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