Министерство образования Республики Беларусь

ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра технологий программирования

Методические указания для выполнения лабораторной работы № 1 по курсу «Операционные системы и системное программирование»

«Разработка программ в ОС UNIX»

Полоцк, 2019

ЦЕЛЬ РАБОТЫ

Работа в командной строке Linux и оболочке "bash". Создание и запуск сценариев. Работа с компилятором gcc.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1. *Оболочка Bash*

Добавление ключей

Практически все команды в Linux допускают использование ключей. Вы можете добавлять ключи и к команде "ls", чтобы видоизменять результат её выполнения или влиять на ход выполнения. Ключ упреждается чертой (например, "ls -a"). Проверьте нижеуказанные варианты выполнения команды "ls", чтобы увидеть соответствующие результаты:

ls -l

Выводит подробный перечень файлов каталога. Для каждого файла или каталога также выводятся владелец, группа, размер, дата изменения, права.

ls -a

Выводит перечень **всех** файлов в каталоге, включая скрытые. В Linux файлы, имена которых начинаются с точки, обычно не отображаются.

ls -R

Выводит содержимое каждого подкаталога, его подкаталоги и.т.д. (рекурсивно).

Если необходимо указать большее количество ключей, Вы можете сгруппировать их через одну черту. Например, команды "ls -al" и "ls -a -l" полностью идентичны.

Некоторые ключи состоят из слова (или слов), и их ввод требует двух черт, а не одной. Например, команда "ls -l --full-time" выводит полную информацию о дате и времени изменения.

Некоторые ключи могут содержать значение. Например, "ls -l --sort=size" сортирует перечень по размеру.

Добавление параметров

Кроме ключей (которые упреждаются одной или двумя чертами), Вы можете устанавливать параметры такие, как имена файлов, каталогов и.т.д.

Например, команда "ls", если Вы не указали каких-либо параметров, выведет содержимое текущего каталога. Но Вы можете добавить параметры, определяющие **что именно** отображать. Например, если ввести "ls /usr", то результатом будет отображение содержимого каталога "/usr". Вы можете указать и большее число параметров.

Получение справочной информации

Команда "man"

Почти каждая команда в Linux имеет доступную через командную строку оперативную справочную информацию. Она вызывается с помощью команды "man" (manual - справочник).

Введите "man ls". Полученная в результате страница описывает работу этой команды, её ключи, другие подробности относительно программы, автора и.т.д. Эта информация отображается с помощью команды "less" (её описание ниже). Для навигации по странице используются клавиши с указанием стрелок, PgUp/PgDn; клавиша Q - выход.

Команда "info"

Другим источником оперативной справочной информации является команда "info". Некоторые команды Linux снабжаются и man -, и info - документацией. Как правило, info - документация более информативна и наглядна, похожа на руководство пользователя, в то время, как man - документация носит справочный характер, указывая перечень ключей, параметров и их описание.

Ввведите "info ls". Для навигации по странице используются те же клавиши, что и для команды "man". Основное отличие заключается в том, что info - страницы могут содержать "меню" ссылок на другие страницы. Чтобы воспользоваться ссылкой, переместите на неё курсор клавишами с указанием стрелок и нажмите Enter.

Ключ "--help"

Большинство (но не все) программ поддерживают ключ --help, который отображает очень краткое описание главных ключей и параметров. Введите "ls --help". Результатом выполнения будет порождение нескольких полноэкранных страниц информации, и Вам придётся воспользоваться линейкой прокрутки для её просмотра.

Ключ "--help" почти не используется, т.к. выводимая им информация редко дополняет man - документацию. Исключение составляет незначительное количество программ, не снабжённых документацией иного рода.

Каталоги Linux

Работая в графическом режиме, в Linux каталоги (другое название - "папки") используют косую черту (/) в качестве разделителя (Windows использует обратную косую черту (\)).

Любой каталог, начинающийся с косой черты, например, "/usr/bin", означает, что это его "*абсолютное*" имя, определяющее полную последовательность каталогов от "корневого" (/) до требуемого (bin). Т.о., когда Вы определяете имя, не имеет значения, какой каталог является текущим, он всегда будет указывать на каталог /usr/bin.

С другой стороны, каталог, начинающийся не с косой черты, является *относительным* к текущему каталогу. Например, "bin" будет указывать на разные каталоги, в зависимости от того, находитесь ли Вы в корневом каталоге (в этом случае он укажет на "/bin"), в каталоге "/usr" (в этом случае он укажет на /usr/bin) или в каталоге "/usr/local" (в этом случае он укажет на /usr/local/bin).

Эти утверждения относятся и к файлам - если Вы указываете "file.txt", то, предположительно, к текущему каталогу, в то время, как "/tmp/file.txt" всегда будет указывать на файл "file.txt" именно во временном каталоге.

Два определённых каталога, находящиеся в текущем каталоге, представляются одной точкой, а родительский каталог представляется двойной точкой. Т.о., если Вы находитесь в каталоге /home/sandbox и введёте ls ..,

результатом будет вывод содержимого родительского каталога, в данном случае /home.

Некоторые системные каталоги

Ниже приведён перечень некоторых общих каталогов для Linux и других Unix - систем, а также их назначение.

/

Обозначает корневой каталог, внутри которого размещаются все остальные каталоги.

Аналогом в Windows является корневой каталог дискового устройства (C:\), но в Linux даже разные дисковые устройства находятся в едином корневом каталоге.

/bin

Сокращённо от "binary - двоичный", содержит программные (исполняемые) файлы. В этом (и других каталогах "bin") находятся команды, такие, как "ls".

В Windows каталог c:\windows\command содержит лишь некоторые консольные программы, остальные разбросаны по другим каталогам.

/dev

Сокращение от "devices - устройства". Содержит определённое количество специальных псевдо-файлов, использующихся для доступа к аппаратной части, а также к подключённым к компьютеру устройств. Например, параллельный порт - это файл с именем "lp0", жёсткий диск - "hda", а его первый раздел - "hda0".

Windows/DOS использует похожий метод, однако в Windows эти файлы не вынесены в отдельный каталог. При попытке доступа к устройствам с именами LPT1, COM1 или CON из любого каталога, Вы получите, соответственно, свойства принтера, подключённого к параллельному порту, последовательного порта или консоли.

/etc

Каталог, в котором хранится почти вся информация о конфигурации в масштабе всей системы. Информация хранится в виде текстовых файлов, т.о. Вы можете просмотреть её с помощью любого текстового редактора. Хотя некоторые файлы являются скрытыми.

Аналогов в Windows нет, т.к. конфигурационные данные, реестр, INI файлы и.т.д., хранятся в различных каталогах.

/home

Домашние каталоги пользователей. Т.о., если Вы создали учётную запись пользователя "sandbox", в этом каталоге создаётся новый подкаталог с именем "sandbox". Пользователям с другими именами он недоступен. Исключением является пользователь, наделённый правами администратора.

Ближайшим аналогом в Windows является каталог c:\windows\profiles, в котором содержатся данные пользователей, а также каталог c:\My Documents, в который сохраняются все документы, созданные пользователем. Другие данные могут быть записаны в других каталогах.

/lib

Каталог, в котором хранятся файлы библиотек. Библиотеки - файлы, содержащие многократно используемые программами функции и процедуры.

Аналогов в Windows/DOS нет.

/mnt

Каталог, в котором хранятся все подмонтированные устройства, а не только жёсткие диски. Обычно содержит подкаталоги "cdrom", "floppy", и.т.д., которые отображают содержимое **подмонтированых** CD-ROM или FDD. Ваши Windows устройства также могут автоматически подмонтироваться в этот каталог.

Аналогов в Windows/DOS нет.

/opt

Каталог, в который устанавливаются дополнительные компоненты системы. Такие продукты, как KDE, Gnome и Oracle могут быть установлены в этот каталог.

Ближайшим аналогом в Windows является каталог c:\Program Files.

/tmp

Каталог, в котором хранятся временные данные. Все файлы, помещённые в него, со временем автоматически удаляются.

Аналог в Windows/DOS - c:\windows\temp.

/usr

Содержит копии большинства корневых каталогов. Например, каталог "bin", содержащий программы, каталог "lib", содержащий библиотеки и.т.д. Обычно "ядровые" файлы Linux содержатся в корневых каталогах, в то время, как "неядровые" файлы находятся в подкаталогах "/usr".

Аналогов в Windows/DOS нет.

/var

Сокращённо от "various - разное". Среди хранящихся здесь файлов системные журналы, спул-файлы (файл, в который в процессе спулинга сбрасывается содержимое задания на печать) и другие файлы *данных*. Спулинг - процесс обработки посылаемых на печать документов, которые сохраняются на диске до момента, когда принтер сможет их обработать.

Аналогов в Windows/DOS нет.

Команды управления каталогами

Здесь описывается большинство общих команд управления каталогами:

mkdir новое-имя-каталога

Создаёт новый каталог с именем "новое-имя-каталога"

cd имя каталога

Перейти к указанному каталогу, делая его "текущим"

cd

Если Вы не указываете имя каталога, переход осуществляется к Вашему "домашнему" каталогу.

rmdir имя-каталога

Перемещает (удаляет) каталог. В качестве меры безопасности каталог должен быть пустым до удаления.

pwd

Отображает текущий каталог.

ls имя-каталога

Выводит содержимое указанного каталога.

Ниже приведена последовательность действий, демонстрирующая практическое применение описанных выше команд. Для наглядности приглашение на ввод команд сделано серым цветом. После отображения содержимого каталога

был создан новый каталог "testing", а затем снова отображено содержимое каталога. Далее мы входим в новый каталог, отображаем текущий каталог, возвращаемся в "домашний" каталог, вновь отображаем текущий каталог. Удаляем каталог "testing".

```
sandbox@laptop:~ > ls
KDesktop  public_html
sandbox@laptop:~ > mkdir testing
sandbox@laptop:~ > ls
KDesktop  public_html  testing
sandbox@laptop:~ > cd testing
sandbox@laptop:~/testing > pwd
/home/sandbox/testing
sandbox@laptop:~/testing > cd
sandbox@laptop:~ > pwd
/home/sandbox
sandbox@laptop:~ > rmdir testing
sandbox@laptop:~ >
```

Команды управления файлами

Здесь описываются большинство общих команд управления файлами.

cp имя-файла1 имя-файла2

cp имя-файла1 имя-файла2 имя-файла3 (и.т.д.) каталог

Копирует файл из **имя-файла1** в **имя-файла2** или (вторая строка) копировать один или более файлов в указанный каталог. **Внимание**: если в указанном месте файл уже существует, он будет перезаписан.

mv имя-файла1 имя-файла2

Переименовывает файл из **имя-файла1** в **имя-файла2**. **Внимание**: если второй файл уже существует, он будет перезаписан.

mv имя-файла1 имя-файла2 имя-файла3 (и.т.д.) каталог

Перемещает один или более файлов в указанный каталог. **Внимание**: если каталог уже содержит файлы с такими именами, они будут перезаписаны.

less имя-файла

Выводит на экран содержимое указанного файла с возможностью навигации по нему клавишами с указанием стрелок, PgUp/PgDn и.т.д. (см. команду "man").

file имя-файла

Отображает идентификатор типа файла, исследуя его содержимое с очень высокой степенью точности.

locate имя-файла-или-каталога

Ищет файл или каталог на указанном диске и отображает все места, где он был найден. Вы можете указать частичное имя или часть полного пути.

Групповые символы

В Linux Вы повсеместно можете указать имя файла и каталога, используя групповые символы. По одному или более специальным символам оболочка будет искать подходящие по шаблону файлы, и помещать их в командную строку вместо шаблона. Слово "wildcard - групповой символ" также переводится как карточный "Джокер", т.к. эта карта может заменить любую другую карту. В данном случае

значок "группового символа" может заменить любые буквы или символы в имени файла.

Проверка групповых символов

Наилучшей тренировкой будет использование команды "ls" в каталоге со множеством файлов с применением групповых символов в качестве аргументов. Как мы видели ранее, команда "ls" может содержать параметр, указывающий, что именно нужно отобразить. Вместо указания каталога мы собираемся передать перечень всех имён файлов, которые необходимо отобразить. Перечень будет составлен согласно указанным шаблонам групповых символов. Их описание приведено ниже.

Перед тем, как продолжить, в окне терминала введите команду "cd /usr/bin". Результатом её выполнения будет переход в главный каталог, содержащий команды ОС. В нём огромное число файлов, т.е. он является идеальной средой для наших экспериментов.

```
sandbox@laptop:~ > cd /usr/bin
sandbox@laptop:/usr/bin >
```

*Групповой символ **

Первый групповой символ - звёздочка. Знак * заменяет нуль и большее количество других символов. Размещая этот символ в начале, в середине или в конце шаблона, Вы можете сформировать конкретный тип задачи. Например, шаблон "*txt" означает любую последовательность букв, оканчивающуюся на "txt".

Ниже представлена таблица шаблонов, примеры соответствующих и несоответствующих этим шаблонам имён файлов.

Ш ы	Соответствующие файлы	Несоответствующие файлы	Причина есоответствия
*	File.txt -file.txt	File.TXT 2	TXT - верхний р ивается на "2" - не в конце ки (.)
*	File.txt -file.txt	File.TXT 2	TXT - верхний р ивается на "2"

			ие в конце
*	File.txt -file.txt 2	File.TXT	TXT - верхний р

Групповой символ ?

В то время, как групповой символ * может заменить *нуль или более* букв или знаков, групповой символ ? заменяет *только один*. Т.о., шаблон "???" заменяет имя файла, состоящее исключительно из трёх символов. Шаблон "x??" означает любое трёхбуквенное имя файла, начинающееся на "x".

Групповой символ [ ]

Квадратные скобки используются для содержания соответствующего *набора* символов. Например, шаблон "[ABC]*" соответствует любому имени файла, начинающегося на одну из букв A, B или C и сопровождающегося любым количеством символов.

Если первый символ - знак восклицания (!) или вставки (^), то шаблон соответствует любому символу, *за исключением* данных. Т.о., шаблон "[^x]*" означает любое имя файла, исключая начинающееся на "x".

Набор может содержать *диапазон* символов, а не только отдельные буквы. Например, шаблон "[A-Z]*" означает любое имя файла, начинающееся с прописной буквы от A до Z включительно, сопровождающегося любым количеством символов, а "[A-Za-z123]" означает один символ, являющийся прописной или строчной буквой или цифрой 1, 2, 3.

Как работают групповые символы

Обработка групповых символов в Windows существенно отличается от их обработки в Linux или других Unix - системах. В Windows выполняемая программа или команда получает выражение групповых символов целостно. Если программа не предназначена для обслуживания групповых символов, она будет пытаться открыть файл, именуемый "*.txt".

С другой стороны, в Linux всю работу выполняет оболочка bash. Она берёт шаблон, содержащий групповые символы, преобразует его в перечень соответствующих имён файлов и передаёт его программе на место шаблона. Ниже приведена таблица, показывающая, как определённые команды будут "транслироваться" оболочкой bash. Фактические имена файлов зависят от содержимого каталога, так что они могут варьироваться.

ИСХОДНАЯ КОМАНДА	ЧТО ФАКТИЧЕСКИ ВЫПОЛНЯЕТСЯ
ls	ls
ls y*	ls yacc ybmtopbm yes ypcat ypchfn ypchsh ypmatch yppasswd ch yuvsplittoppm yuvtoppm
ls ?a?	ls cal man tac

ls blubble*	ls blubble*

Обратите внимание на последний пример - оболочка bash не смогла найти файл с таким именем, т.о., шаблон, содержащий групповые символы, передаётся программе в форме "как есть".

Это "облегчает жизнь" программам, т.к. они должны обслужить только списки имён файлов на их командных строках. Однако есть несколько приёмов, которые можно сделать в MS-DOS, и нельзя в Linux. Например, в MS-DOS можно ввести команду "copy *.doc *.bak" - которая скопирует все файлы с расширением "doc" в одноимённые файлы с расширением "bak". В Linux эта команда будет переведена в нечто похожее на "copy file1.doc file2.doc file3.doc file2.bak file4.bak" - что даст совершенно другой и, возможно, нежелательный результат. Из-за введения длинных имён файлов эта методика фактически не работает в Windows.

Работа с каталогами при помощи групповых символов

В Linux групповыми символами можно управлять и каталогами. Например, шаблон "*/file.txt" означает все файлы "file.txt" в любом подкаталоге.

Работа со скрытыми файлами при помощи групповых символов

Групповые символы не работают со скрытыми файлами, если шаблон группового символа начинается без точки. Т.о., шаблон ".*" соответствует всем скрытым файлам. Скрытые файлы - это файлы, имя которых начинается с точки. Например, .profile или .kde2.

Возможности ввода команд

Работая в оболочке bash, Вы можете *вызывать* ранее введённые команды с помощью клавиш ↑ и ↓.

Существуют также bash_history (история bash) и комбинация Ctrl-R, позволяющая быстро вызывать команды из этой истории по ключевым последовательностям символов. Т.о. если Вы хотите найти последнюю смену каталога, введите "[Ctrl-R]cd", и в командной строке отобразится команда "cd".

При вводе первых символов имени файла или каталога, нажмите [Tab]. Оболочка bash автоматически заполнит его, при условии, что такой файл существует и начинается с введённых Вами символов. Например, если Вы ввели "ls br[Tab]", оболочка bash заполнит имя файла до "brushtopbm", при условии, что такой файл существует и что только он начинается на "br".

Перенаправление вывода в файл

Большинство программ при своём выполнении выводят на экран много текста. Вы можете сохранить этот текст в файл для возможного дальнейшего использования. Чтобы это сделать, воспользуйтесь оператором перенаправления ">".

Например, Вы хотите сохранить копию содержимого каталога в файл. Вы вводите команду "ls -l" с последующим ">", а также имя создаваемого файла.

В нижеприведённом примере мы отобразим содержимое каталога /usr в подробном формате, а затем запишем его в файл в наш "домашний" каталог (т.к. мы не можем создавать файлы в каталоге /usr).

```
sandbox@laptop:~ > cd /usr
sandbox@laptop:/usr > ls -l > ~/usr-listing.txt
```

```
sandbox@laptop:/usr > cd
sandbox@laptop:~ > ls
KDesktop  public_html  snapshot1.png  usr-listing.txt
sandbox@laptop:~ >
```

Обратите внимание на вторую строку. Сначала мы дали команду вывода содержимого каталога "ls -l", затем оператором перенаправления ">" указали оболочке сохранить результаты в файл с именем listing.txt (~/usr-listing.txt). Символ "~" обозначает "домашний" каталог пользователя для возможного повторного быстрого вызова.

Приведённый символ (>) *создаёт* новый файл с указанным именем. Если такой файл уже существует, он очищается перед получением вывода программы. Если Вы вводите двойной символ (>>), вывод программы будет *дописываться* в конец существующего файла. Это нужно при перенаправлении выводов нескольких программ в один файл.

Конвейеризация вывода в программу

В предыдущей главе мы рассмотрели перенаправление вывода команды в новый файл. Вывод одной программы можно передать на вход другой программы, применяя конвейеризацию. Вторая программа обрабатывает вывод первой программы, на основе которого порождает свой вывод.

Проверим. Содержимое файла отображается командой "cat". Файл "/etc/services" содержит список распознанных TCP/IP сервисов. Отображение занимает несколько полноэкранных страниц текста.

Учтите, что в нижеприведённом примере показан фрагмент отображаемой информации.

```
sandbox@laptop:~ >cat /etc/services
#               0/tcp    Reserved
#               0/udp    Reserved
tcpmux          1/tcp              # TCP Port Service Multiplexer
tcpmux          1/udp               # TCP Port Service Multiplexer
compressnet      2/tcp              # Management Utility
compressnet      2/udp               # Management Utility
.....множество других строк.....
nimhub         48002/tcp            # Nimbus Hub
nimhub         48002/udp             # Nimbus Hub
nimgtw         48003/tcp            # Nimbus Gateway
nimgtw         48003/udp             # Nimbus Gateway
```

Теперь команда "sort" сортирует ввод в алфавитном порядке и передаёт его на вывод. Т.о., для сортирования вывода команды "cat", мы должны *конвейеризировать* его на вход команды sort. Чтобы это сделать, нужно воспользоваться символом конвейеризации (|).

```
sandbox@laptop:~ >cat /etc/services | sort
3Com-nsd        1742/tcp             # 3Com-nsd
3Com-nsd        1742/udp              # 3Com-nsd
3com-amp3        629/tcp             # 3Com AMP3
```

```
3com-amp3      629/udp            # 3Com AMP3
.....множество других строк.....
zip            6/ddp              # Zone Information Protocol
zserv          346/tcp             # Zebra server
zserv          346/udp             # Zebra server
```

Одно очень популярное применение конвейера заключается в передаче вывода таких длинных команд на вход команды "less". Эта команда позволяет полностью пролистать полученную информацию.

```
ps -Hefw | less
```

Указанная командная строка отображает список системных процессов, а применение команды "less" позволяет полностью пролистать полученную информацию, используя клавиши с указанием стрелок, PgUp/PgDn; клавиша Q - выход.

Возможен одновременный ввод нескольких команд, разделённых знаком |. Например, мы можем использовать команду "cat" для отображения содержимого файла сервисов, передать результат на вход команды "sort" для сортировки, отсортированный список подать на вход команды "tail" для выделения последних 50 строк и, наконец, передать эти 50 строк на вход команды "less" для просмотра полученных результатов.

```
cat /etc/services | sort | tail -n 50 | less
```

2. ***Сценарии***

Сценарий оболочки - это файл, содержащий серию приведённых выше команд, которые будут последовательно выполнены оболочкой bash. Технически это программа, написанная на языке "bash".

Для создания сценариев можно использовать любой текстовый редактор. В KDE можно использовать любой расширенный редактор - kate, kwrite или kedit. **Не следует** применять программы подготовки текста - они не являются текстовыми редакторами, и добавляют в текст дополнительное форматирование. Если Вы знаете, как создать в них беспримесный текстовый файл, пользуйтесь ими.

Создайте новый файл и введите следующие строки.

```
#!/bin/bash
echo "I am about to list the home directory"
# here is the actual listing:
ls $HOME
echo "Done!"
```

Первая строка указывает текущей оболочке, какую программу следует использовать для интерпретации файла. В данном случае это оболочка bash. Это сделано для того, что если сценарий вызывается в пределах другой оболочки или файлового менеджера - Konqueror и Nautilus, они будут "знать", что для выполнения этого сценария требуется оболочка bash.

Вторая строка - это первая команда в сценарии. Команда "echo" используется для вывода на экран простой информационной строки.

Третья строка - комментарий. Оболочка bash проигнорирует его, но для пользователя он полезен, т.к. поясняет ход работы сценария. В простых сценариях комментарии необязательны, но вот в сложных они уместны.

Далее следуют ещё две команды - команда "ls", которая берёт имя каталога в качестве параметра, и, наконец, команда "echo", выводящая на экран информацию об успешном выполнении работы!

При сохранении этого файла оказывается, что он неисполняемый. Чтобы сделать его исполняемым, нужно использовать команду "chmod". В качестве примера я назвал этот сценарий "myscript" - Вы можете назвать его по другому.

```
chmod u+x myscript
```

Теперь Вы можете запустить на выполнение новый сценарий. Для этого введите следующую команду, заменив имя сценария "myscript" на то, которое выбрали Вы.

```
sandbox@laptop:~ >./myscript
I am about to list the home directory
KDesktop  myscript  public_html  snapshot1.png  usr-listing.txt
Done!
```

Учтите, что "./" перед именем файла - это актуальное имя каталога (точка означает текущий каталог). Можно воспользоваться переменной окружения PATH для того, чтобы указать каталог, в котором находятся созданные Вами сценарии. Теперь Вы можете вводить команды независимо от того, в каком каталоге Вы находитесь.

Сценарии автозапуска Bash

Оболочка bash использует специальный, скрытый файл, находящийся в Вашем домашнем каталоге: ".bashrc". По сути, это тот же сценарий, но в нём отсутствует первая строка, которую мы использовали выше (#!/bin/bash). Этот сценарий автоматически выполняется при запуске оболочки bash. В него можно помещать переменные окружения или другие параметры для того, чтобы они выполнялись оболочкой bash при каждом входе в систему.

Псевдоимена

Оболочка bash позволяет назначать новые команды, заменяющие длинноформатные команды. Например, если ввести...

```
alias lh="ls -l -a $HOME"
```

, то команда "lh" будет эквивалентна команде "lh -l -a $HOME".

Вы можете использовать данную возможность для переопределения существующих команд. Например, если Вы введёте "alias ls="ls -a"", то команда ls всегда будет отображать скрытые файлы. Вы можете удалить существующее псевдоимя командой unalias.

Эти псеводимена Вы можете поместить в файл .bashrc, чтобы они восстанавливались при загрузке оболочки bash.

Для некоторых команд Вы можете добавлять даже "меры обеспечения безопасности". Например, Вы создали псевдоимя для команды копирования:

```
alias cp="cp -i"
```

При каждой операции копирования она будет запрашивать подтверждение, если копируемый файл уже существует, вместо того, чтобы перезаписать его без предупреждения. Учтите, что эти настройки действительны только для учётной записи конкретного пользователя. При входе в систему с использованием учётной записи другого пользователя они работать не будут. Для того, чтобы они работали, Вы должны добавить эти псевдоимена в файл .bashrc данного пользователя.

Переключение в режим администратора

Иногда возникает ситуация, когда для выполнения некоторых команд необходимо переключиться в режим администратора. Помните, что, работая в режиме администратора, Вы можете делать всё - устанавливать файлы в системную область или удалить все данные с жёсткого диска. Будьте внимательны при входе в систему в режиме администратора.

Команда переключения между учётными записями пользователей: su - сокращённо от "set user" (установить пользователя). Ввод этой команды без дополнительных параметров означает переключение в режим администратора. Чтобы переключиться на учётную запись конкретного пользователя, необходимо указать имя этого пользователя. Во всех случаях Вам будет предложен ввод пароля. После ввода пароля Вы можете приступить к работе с данными и файлами, которые доступны только для этого пользователя и.т.д. Администратору доступны все файлы и задачи.

Попробуйте начать работу с использованием Вашего обычного имени пользователя. В данном примере я использую 'ramon' в качестве имени пользователя.

```
sandbox@laptop:~ > ls /home/ramon
Access denied.
sandbox@laptop:~ > ls
file1.txt   file2.txt
sandbox@laptop:~ > su ramon
Password: (ввод пароля для ramon)
ramon@laptop:/home/sandbox > ls
Access denied.
ramon@laptop:/home/sandbox > ls /home/ramon
Desktop  Mail  Documents
ramon@laptop:/home/sandbox > exit
sandbox@laptop:~ >
```

Учтите, что, войдя в систему под именем "sandbox", я могу отобразить содержимое домашнего каталога пользователя "sandbox", домашние каталоги других пользователей недоступны. Обратите также внимание, что я воспользовался командой exit для возврата в учётную запись "sandbox" из записи "ramon". Это необходимо, т.к. команда su создаёт новый процесс оболочки, используя учётную запись "ramon".

При использовании команды su без дополнительных параметров выдаётся приглашение на ввод пароля администратора. Войдя в систему в режиме администратора, пользователю доступны все области.

```
sandbox@laptop:~ > su
Password: (ввод пароля администратора)
root@laptop:/home/sandbox > ls
file1.txt   file2.txt
root@laptop:/home/sandbox > ls /home/ramon
Desktop  Mail  Documents
root@laptop:/home/sandbox > exit
sandbox@laptop:~ >
```

Существует различие между входом в систему в режиме администратора (или другого пользователя) с дальнейшей загрузкой терминала и входом в режиме обычного пользователя с последующими загрузкой терминала и использованием команды su для переключения в режим администратора. Различие заключается в загружаемых сценариях автозапуска. При описанном выше использовании команды "su" она наследует все установки от предыдущей оболочки. Т.о., например, если Вы установили некоторые переменные окружения, они будут перенесены и доступны при работе в режиме администратора. Это полезно при некоторых компиляциях. Если Вы компилируете программу в режиме обычного пользователя, а затем переключаетесь в режим администратора для окончательного установочного шага, может оказаться полезным перенос некоторых установленных ранее переменных окружения.

Существует один недостаток - некоторые команды в сценариях автозапуска администратора не будут выполняться. Если Вам нужно переключиться в режим администратора (или другого пользователя) и быть уверенным, что сценарии автозапуска выполнятся, добавьте дефис между командой su и параметром имени пользователя. Т.о., применённая выше команда будет иметь такой вид:

```
su - ramon
su -
```

Вам снова будет предложено ввести пароль, но теперь сценарии автозапуска будут выполнены, а любые установки Вашего сеанса будут "забыты" - так, словно Вы вошли в систему как пользователь, имя которого отображено на экране.

Компиляция программ из исходных кодов

Одной из основных причин использования терминала является компиляция программ, скачанных из Интернета. Компиляция - это процесс, преобразующий исходный код в программу в двоичном представлении с учётом типа процессора, установленного на компьютере.

Все подобные программы сопровождаются инструкцией по компиляции - обычно это текстовые файлы с именами "INSTALL", "README" и.т.п. Ниже описывается общий для всех программ метод.

Допустим, что скачанный файл называется "MyProgram-1.2.3.tgz". Т.к. это файл - архив, наподобие WinZip файла, необходимо извлечь его содержимое. Это делается с помощью команды tar:

```
tar -xvzf MyProgram-1.2.3.tgz
```

Приведенные ключи указывают команде tar извлечь (x) и подробно описать (v) сжатое при помощи архиватора ZIP содержимое (z) архивного файла (f) с указанным именем. Tar извлечёт содержимое архива и отобразит что оно делает.

Далее необходимо перейти в только что созданный каталог, имеющий такое же имя, что и архив, но без расширения ".tgz".

```
cd MyProgram-1.2.3
```

Следующий шаг - запуск сценария "configure". Он проверяет Вашу систему и порождает компилирование и установочный сценарий, идеально подходящий для Вашей конфигурации. Другая задача на этом этапе заключается в проверке установленного необходимого программного обеспечения. Команда "configure" по окончанию своей работы выводит большой объём информации. Здесь она не приводится, т.к. обычно всё проходит без ошибок.

```
./configure
```

В то время, как сценарий "configure" пишет рецепт, команда "make" берёт ингредиенты и печёт пирог. Обычно это делается в два этапа. Сначала Вы вводите "make" для генерации программы в двоичном представлении, затем "make install" для размещения этой программы в надлежащем месте Вашего компьютера. Последний шаг должен выполняться в режиме администратора, т.к. он требует разрешение разместить файлы в системных областях. Переключиться в режим администратора Вы можете при помощи команды "su".

```
make
su
make install
exit
```

Если сообщений об ошибках не было, значит, всё прошло успешно.

Передача параметров

Аргументы, передаваемые скрипту из командной строки -- $0, $1, $2, $3..., где $0 -- это название файла сценария, $1 -- это первый аргумент, $2 -- второй, $3 -- третий и так далее. Аргументы, следующие за $9, должны заключаться в фигурные скобки, например: ${10}, ${11}, ${12}.

Специальные переменные $* и $@ содержат *все* позиционные параметры (аргументы командной строки).

Скобочная нотация позиционных параметров дает довольно простой способ обращения к *последнему* аргументу, переданному в сценарий из командной строки.

```
args=$#          # Количество переданных аргументов.
lastarg=${!args}  # Обратите внимание: lastarg=${!$#} неприменимо.
```

3. ***Компилятор GCC***

Gcc - это свободно доступный оптимизирующий компилятор для языков C, C++, Ada 95, а также Objective C. Его версии существуют для различных реализаций

Unix (а также VMS, OS/2 и других систем PC), и позволяют генерировать код для множества процессоров.

Вы можете использовать gcc для компиляции программ в объектные модули и для компоновки полученных модулей в единую исполняемую программу. Компилятор способен анализировать имена файлов, передаваемые ему в качестве аргументов, и определять, какие действия необходимо выполнить. Файлы с именами типа name.cc (или name.C) рассматриваются, как файлы на языке C++, а файлы вида name.o считаются объектными (т.е., внутримашинным представлением).

Чтобы откомпилировать исходный код C++, находящийся в файле F.cc, и создать объектный файл F.o, выполните команду:

gcc -c <compile-options> F .cc

Здесь строка compile-options указывает возможные дополнительные опции компиляции.

Чтобы скомпоновать один или несколько объектных файлов, полученных из исходного кода C++ - F1.o, F2.o, ... - в единый исполняемый файл F, используйте команду:

gcc -o F <link-options> F1.o F2.o ... -lg++ <other-libraries>

Здесь строка link-options означает возможные дополнительные опции компоновки, а строка other-libraries - подключение при компоновке дополнительных разделяемых библиотек.

Вы можете совместить два этапа обработки - компиляцию и компоновку - в один общий этап с помощью команды:

gcc -o F <compile-and-link-options> F1.cc ... -lg++ <other-libraries>

После компоновки будет создан исполняемый файл F, который можно запустить с помощью команды

./F <arguments>,

где строка arguments определяет аргументы командной строки Вашей программы.

В процессе компоновки очень часто приходится использовать библиотеки. Библиотекой называют набор объектных файлов, сгруппированных в единый файл и проиндексированных. Когда команда компоновки обнаруживает некоторую библиотеку в списке объектных файлов для компоновки, она проверяет, содержат ли уже скомпонованные объектные файлы вызовы для функций, определенных в одном

из файлов библиотек. Если такие функции найдены, соответствующие вызовы связываются с кодом объектного файла из библиотеки.

Библиотеки обычно определяются через аргументы вида -llibrary-name. В частности, -lg++ означает библиотеку стандартных функций C++, а -lm определяет библиотеку различных математических функций (sin, cos, arctan, sqrt, и т.д.). Библиотеки должны быть перечислены после исходных или объектных файлов, содержащих вызовы к соответствующим функциям.

Среди множества опций компиляции и компоновки наиболее часто употребляются следующие:

-c

Только компиляция. Из исходных файлов программы создаются объектные файлы в виде name.o. Компоновка не производится.

-Dname=value

Определить имя name в компилируемой программе, как значение value. Эффект такой же, как наличие строки #define name value в начале программы. Часть `=value' может быть опущена, в этом случае значение по умолчанию равно 1.

-o file-name

Использовать file-name в качестве имени для создаваемого gcc файла (обычно это исполняемый файл).

-llibrary-name

Использовать при компоновке указанную библиотеку.

-g

Поместить в объектный или исполняемый файл отладочную информацию для отладчика gdb. Опция должна быть указана и для компиляции, и для компоновки.

-MM

Вывести заголовочные файлы (но не стандартные заголовочные файлы), используемые в каждом исходном файле, в формате, подходящем для утилиты make. Не создавать объектные или исполняемые файлы.

-pg

Поместить в объектный или исполняемый файл инструкции профилирования для генерации информации, используемой утилитой gprof. Опция должна быть указана и для компиляции, и для компоновки. Профилирование - это процесс измерения продолжительности выполнения отдельных участков Вашей программы. Когда Вы указываете -pg, полученная исполняемая программа при запуске генерирует файл статистики. Программа gprof на основе этого файла создает расшифровку, указывающую время, потраченное на выполнения каждой функции.

-Wall

Вывод сообщений о всех предупреждениях или ошибках, возникающих во время трансляции программы.

-O1

Оптимизация уровня 1. Оптимизированная трансляции требует несколько больше времени и несколько больше памяти для больших функций. Без указания опций `-O' цель компилятора состоит в том, чтобы уменьшить стоимость трансляции и выдать ожидаемые результаты при отладке. Операторы независимы: если вы останавливаете программу на контрольной точке между операторами, вы можете назначить новое значение любой переменной или поставить счетчик команд на любой другой оператор в функции и получить точно такие результаты, которые вы ожидали из исходного текста. С указанием `-O' компилятор пробует уменьшить размер кода и время исполнения.

-O2

Оптимизация уровня 2. GNU CC выполняет почти все поддерживаемые оптимизации, которые не включают уменьшение времени исполнения за счет увеличения длины кода. Компилятор не выполняет раскрутку циклов или подстановку функций, когда вы указываете `-O2'. По сравнения с `-O' эта опция увеличивает как время компиляции, так и эффективность сгенерированного кода.

-O3

Оптимизация уровня 3. `-O3' включает все оптимизации, определяемые `-O2', а также включает опцию `inline-functions'.

-O0

Не оптимизировать. Если вы используете многочисленные `-O' опции с номерами или без номеров уровня, действительной является последняя такая опция.

ЗАДАНИЯ ДЛЯ ЛАБОРАТОРНЫХ РАБОТ

1. Изучить команды оболочки bash;
2. Научиться создавать сценарии;
3. Изучить команды утилиты gcc;
4. Выполнить следующие задания:

№	Задание
1	Вычисление нескольких математических выражений из трех операндов, каждое математическое выражение в отдельном пакетном файле. Номер выражения и три операнда передаются как параметры главного пакетного файла. Результат и математическое выражение вывести на экран. (Варианты см. ниже)
2	Написать простейшую программу на языке Си которая выводит фамилию студента, скомпилировать и скомпоновать ее утилитой gcc.

Варианты для первого задания:

1	2 – 5 + 3, 77 * 0 + 6, 1 – 1 - 1
2	28 – 6 * 7, 5 / 1 + 5, 88 * 0 - 4
3	6 + 8 / 4, 5 + 5 – 5, 2 * 64 / 2
4	7 + 7 * 0, 9 – 4 / 2, 1 – 1 + 3
5	2 * 8 / 16, 4 + 5 – 9, 20 / 5 /4

СОДЕРЖАНИЕ ОТЧЕТА

1. Ф.И.О., группа, название лабораторной работы.
2. Цель работы.
3. Описание проделанной работы.
4. Результаты выполнения лабораторной работы.
5. Выводы.