ПРАКТИКУМ

по курсу

ОПЕРАЦИОННЫЕ СИСТЕМЫ. КОМАНДНЫЙ ИНТЕРФЕЙС UNIX.

Содержание:

Оглавление

Введение

Практикум преследует цель закрепления у студентов начальных сведений о командном языке операционных систем семейства UNIX и основных операторах языка интерпретатора shell. Практикум может быть выполнен в среде любой доступной операционной системы (System V, AIX, Linux и др.) в многотерминальном режиме. В качестве базового принят Bash-shell. Команды этой оболочки входят в стандарт POSIX, т.е. являются принадлежностью любой UNIX-подобной операционной системы. При сдаче практикума преподавателю будет использоваться операционная система Debian.

Практикум рассчитан на студентов очной, очно-заочной и заочной форм обучения бакалавров по направлениям 230100 «Информатика и вычислительная техника», 010400 «Прикладная математика и информатика», 230700 «Прикладная информатика в экономике», а также всем изучающим основы операционных систем семейства UNIX.

Порядок выполнения практикума:

1. Выполнить все задания лабораторной работы (выполнение заданий должно подтверждаться выводом команды **history**).
2. Подготовить ответы на все контрольные вопросы для защиты лабораторной преподавателю. Преподаватель может задавать любые вопросы, относящиеся к теме лабораторной работы.

Практикум считается выполненным, если сданы все лабораторные работы.

Синтаксис команд интерпретатора можно представить в следующем обобщенном виде:

$ имя_команды [-ключи] [аргумент [аргументы]] <Enter>

Приглашение $ и управляющая клавиша <Enter> необходимы для синхронизации работы операционной системы и пользователя. Квадратные скобки ([]) в записи команды указывают на необязательные параметры, угловые скобки (<>) – на обязательные параметры. Скобки используются только при описании синтаксиса команд и не вводятся при их выполнении. Символ | означает несколько возможных вариантов, а многоточие (...) - то, что параметр может повторяться.

Каждый из пользователей перед началом работы должен получить имя и пароль своей учетной записи в системе у администратора, тем самым ему разрешается использование ресурсов системы и выделяется личный каталог.

Пароль может быть назначен администратором или изменен самим пользователем с помощью специальной команды:

$ passwd [входное_имя]

Протокол работы с системой при выполнении практикума:

```
Login: <набор_лог.имени>      <Enter>
Password: <набор_пароля>      <Enter>
 [представление системы]
$ <ввод_команды>              <Enter>
[сеанс работы с системой]
<Ctrl*D> или exit             <Enter>
```

Командой **exit** необходимо завершать сеанс работы с системой, так как только при этом завершаются все процессы, обслуживавшие данный терминал пользователя.

При выполнении практикума вы можете пользоваться следующими клавишами для быстрого выполнения команд:

- Клавиша «Вверх» — просмотр предыдущей выполненной команды.
- Shift + PageUp — прокрутить экран вверх.
- Shift + PageDown — прокрутить экран вниз.
- Tab — дополнить название команды или файла, начинающееся с введенных букв.
- Двойной Tab — вывести доступные названия команд и/или файлов, начинающиеся с введенных букв.
- Ctrl + Insert — копировать выделенный текст.
- Shift + Insert — вставить выделенный текст.
- Ctrl + R — поиск по истории выполненных команд.

Практикум предусматривает выполнение учащимися до пяти лабораторных работ и требует в общей сложности не менее 10 двухчасовых занятий на компьютере.

Для получения информации о порядке использования команд, изучаемых в ходе выполнения лабораторного практикума, используйте команду **man**:

$ man команда

Если команда встроена в оболочку (вместо руководства по команде открывается справочная информация по bash), информацию о ней можно получить, используя команду **help**:

$ help команда

Для анализа проделанной лабораторной работы в целом и подготовки её к зачету можно использовать команду **history** — вывод на экран выполненных команд.

Практикум может выполняться в дистанционном режиме. Для этого необходимо подключиться к серверу по адресу **samos.dozen.mephi.ru** по протоколу SSH, используя свои

имя пользователя и пароль. В среде Windows для этого используется приложение Putty (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html). В среде MacOS и Linux используется встроенная команда **ssh**.

Лабораторная работа №1. Знакомство с ОС UNIX

Основы работы с командами в консоли ОС UNIX

Основные понятия:

- сеанс работы
- виртуальные консоли
- оболочка
- рабочая среда
- удаленная консоль

Используемые команды:

alias	вывод списка или создание нового алиаса для команды
date	вывод или установка системной даты и времени
df	вывод информации об использовании дискового пространства
env	вывод информации о среде, запуск приложений с измененными переменными среды
exit	завершение сеанса работы
hostname	вывод имени машины
man	вывод справки по команде
uname	вывод системной информации
whereis	вывод информации о расположении файла
which	вывод полного пути до файла вызываемого командой

UNIX — многозадачная многопользовательская операционная система. Система может работать в режиме с графической оболочкой или без нее. В рамках курса изучения ОС UNIX все работы будут проводиться в системе, работающей в многопользовательском режиме с поддержкой сети без графической оболочки.

Системная консоль — это монитор и клавиатура, связанные непосредственно с системой. Для входа в систему под несколькими учетными записями, используя один монитор и одну клавиатуру, система обеспечивает доступ к **виртуальным консолям,** которые позволяют войти в систему под несколькими именами в одно время. Сеанс работы начинается со входа в систему, для чего пользователь должен ввести свое имя и пароль. Для завершения сеанса работы с системой вы можете воспользоваться командой **exit** или сочетанием клавиш **CTRL-D.**

Оболочка (shell, командный интерпретатор) — это программа, которая воспринимает введенные пользователем данные и транслирует это в системные команды. Оболочка запускается сразу после входа пользователя в систему. Используя язык обращения к оболочке можно создавать скрипты.

Рабочая среда — это множество переменных, к которым имеют доступ все выполняемые команды.

Файл /etc/profile содержит переменные среды на уровне всей системы. Файл .bash_profile содержит переменные среды пользователя. При входе в систему первыми скриптами, выполняемыми оболочкой, являются эти скрипты.

Задание 1

1. Войдите в систему, используя имя пользователя и пароль, выданные вам преподавателем.
2. Определите имя машины. Найдите две разные команды, которыми можно это сделать.
3. Определите архитектуру процессора машины, используемой вами.
4. Выведите на экран время и дату в формате «31/12/2099 23:59».
5. Выведите на экран полный список алиасов пользователя. Создайте алиас для команды **newdate**, чтобы она выводила дату в формате «31/12/2099 23:59».
6. Выведите на экран переменные среды. Объясните, что означают переменные SHELL, USER, PATH, HOME, LANG, MAIL, PS1.
7. Измените переменную PS1, что изменилось?
8. Запустите приложение **bash**, изменив переменную домашнего каталога пользователя. Проверьте, что изменения вступили в силу. Как это можно сделать?
9. Определите местоположение в системе файла вызываемого командой **bash**, используя команду **which.**

Пользователи и группы

Основные понятия:

- учетная запись пользователя
- домашний каталог
- идентификатор пользователя

Используемые команды:

id	вывод информации о текущем пользователе
groups	просмотр списка групп, участником которых является текущий пользователь
finger	поиск информации о пользователях
chsh	изменение стандартной оболочки пользователя
chfn	изменение информации о пользователе, выводимой командой finger
last	вывод списка пользователей, заходивших в систему в последнее время
mail	отправление и чтение почтовых сообщений
mesg	включение/выключение отправки прямых сообщений на терминал пользователя
passwd	изменение пароля пользователя

who	вывод списка пользователей, находящихся в системе в текущий момент
whoami	имя текущего пользователя
write	отправка прямого сообщения на терминал пользователя

Для идентификации пользователей в системе используются учетные записи. Вводя имя пользователя и пароль при входе в системе, вы представляетесь системе, позволяя ей открыть вашу домашнюю папку, выдать вам права на нужные каталоги и подключить другие специфичные для вас атрибуты. Каждому пользователю соответствует набор сведений:

- системное имя пользователя — имя, которое вы используете при входе в систему;
- идентификатор пользователя (UID) — уникальный номер пользователя в системе;
- идентификатор группы (GID) — номер основной группы, к которой относится пользователь;
- комментарий (как правило, полное имя — имя и фамилия или псевдоним пользователя, по которому другие пользователи могут определить, кому принадлежит учетная запись);
- домашний каталог — личный каталог пользователя, на доступ к которому пользователь имеет полные права;
- начальная оболочка — оболочка, запускаемая системой при входе пользователя в систему.

Пользователи системы состоят в группах, используемых для организации доступа нескольких пользователей к ресурсам. Каждый пользователь состоит минимум в одной группе, название которой, как правило, совпадает с именем пользователя, и которая создается вместе с учетной записью пользователя.

Пользователь может самостоятельно изменить свой пароль для входа в систему, используя команду **passwd.** Изменить начальную оболочку можно, используя команду **chsh.**

С помощью команды **who** можно получить информацию о пользователях, которые вошли в систему. На экране появится примерно такая информация:

```
[elvis@station elvis]$ who
elvis    tty2         May  5 15:07
root     tty1         May  3 07:50
blondie  :0           May  5 08:48
blondie  pts/0        May  5 09:03 (:0.0)
```

В первом столбце отображается список пользователей, которые вошли в систему, в последних столбцах — дата и время входа. Данные второго столбца показывают, откуда пользователь вошел в систему. Первые два пользователя (elvis и root) использовали для входа tty1 и tty2, что означает первую и вторую виртуальную консоль соответственно. :0 для пользователя blondie означает вход в с использованием графического интерфейса, а pts/0 относится к первому из терминалов, открытых в графической среде.

Задание 2

1. Определите имя вашего пользователя.
2. Определите имена пользователей, работающих с системой в данный момент. Найдите себя в этом списке.
3. Определите имена трех последних пользователей кроме вас, заходивших в систему.
4. Определите, в каких группах состоит ваш пользователь.
5. Откройте файл /etc/passwd, используя команду **more /etc/passwd**. Найдите запись с данными вашего пользователя. Проанализируйте все поля записи и опишите их назначение. Используйте справку по файлу (**man 5 passwd**) для ознакомления со значением всех полей.
6. Выведите на экран информацию о пользователе, используя команду **finger.** Попробуйте изменить полное имя пользователя и добавить свой номер телефона при помощи команды **chfn.**
7. Договоритесь с соседним пользователем об организации обмена прямыми сообщениями. Обменяйтесь с ним сообщениями в режиме прямого диалога.
8. Исследуйте возможности блокирования и разблокирования средств приема сообщений.
9. По договоренности с коллегами обменяйтесь несколькими почтовыми сообщениями. Проанализируйте возможности обработки поступивших почтовых сообщений.
10. Проанализируйте с использованием команды history содержание лабораторной работы, продумайте ответы на нижеприведенные контрольные вопросы и сдайте выполненную работу преподавателю. После получения зачета по работе – уничтожьте все созданные файлы и корректно выйдите из системы.

Контрольные вопросы:

1. Опишите процесс загрузки операционной системы.
2. Что такое уровни инициализации, и зачем они нужны?
3. В чем заключается процедура авторизации пользователя? Цель авторизации? Какие действия совершает система после того, как пользователь ввел пароль, и до того, как пользователь получает возможность передавать системе какие-то команды?
4. Опишите две реальные ситуации, когда вам может понадобиться получить дату и время в консоли UNIX.
5. Что такое среда пользователя? Опишите, как добавить новую переменную в среду, как изменить значение существующей переменной для одного пользователя и всех пользователей в системе.
6. Опишите одну реальную ситуацию, когда вам может понадобиться изменить переменную среды при запуске приложения.
7. Для чего используются группы пользователей?
8. Зачем нужны идентификаторы пользователей?
9. Объясните содержание и назначение каждого поля регистрационной записи.
10. В чем отличие в диалоге прямыми сообщениями и почтовыми?

Лабораторная работа №2. Управление файлами и каталогами

Основные понятия:

- Корень каталогов
- Абсолютный путь
- Относительный путь
- Текущий рабочий каталог
- Домашний каталог
- Жесткая ссылка
- Символьная ссылка
- Сломанная символьная ссылка

Используемые команды:

>	создание нового файла или перенаправление потока вывода в файл
cat	вывод содержимого текстового файла
cd	переход в другой каталог
cp	копирование файлов и каталогов
du	вывод информации о месте, занимаемом на диске файлом или каталогом
find	поиск файлов в каталоге
head	вывод первых строк файла
less	вывод содержимого файла на экран с возможностью прокрутки
ln	создание ссылки на файл или каталог
ls	вывод списка файлов и подкаталогов в текущем каталоге
mkdir	создание нового каталога
more	постраничный вывод содержимого файла
mv	перемещение файла или каталог
pwd	вывод пути до текущего каталога
rmdir	удаление каталога
rm	удаление файла
tail	вывод последних строк файла
tree	вывод на экран иерархии каталогов
sort	сортировка строк в файлах и выводе команд
touch	создание нового файла
wc	подсчет количества строк, слов и байт в файле

Все данные, хранимые на диске, представлены в виде **файлов**. Файлы имеют имена, позволяющие пользователям обращаться к данным файла.

Каталог — это совокупность файлов. Каталоги организованы в древовидную структуру.

Текущий рабочий каталог — это каталог, в котором находится пользователь, вызывая ту или иную команду. Узнать текущий рабочий каталог можно, используя команду **pwd.**

Домашний каталог — каталог, в котором хранятся личные файлы пользователя. Каждый пользователь в системе имеет свой личный каталог. В системе ваш домашний каталог обозначается спецсимволом «~», который вы можете использовать для перехода в домашний каталог (**cd ~**), копирования файлов в домашний каталог и в других случаях.

Корневой каталог – первый каталог в древовидной структуре системы, для которого все остальные каталоги являются вложенными. Обратиться к корневому каталогу можно, используя спецсимвол «/». Например, для перехода в корневой каталог необходимо использовать команду **cd /**. Основные каталоги:

- /bin — каталог содержит исполняемые файлы, например, ls, vi, cd, cp.
- /boot — каталог содержит файлы, необходимые для загрузки системы, например ядро linux и файлы загрузчика (lilo, grub или другого).
- /dev — каталог содержит файлы устройств, присоединенных к системе, или файлы виртуальных устройств, созданных ядром.
- /etc — каталог содержит файлы конфигурации большей части программ и приложений.
- /home — каталог содержит домашние каталоги пользователей.
- /lib — каталог содержит библиотеки, необходимые для исполнения приложений из каталогов /bin и /sbin.
- /root — домашний каталог пользователя root.
- /sbin — каталог содержит исполняемые файлы, используемые при загрузке системы или для ее администрирования суперпользователем root.
- /tmp — общий каталог, используемый для хранения временных файлов.
- /usr — каталог содержит пользовательские приложения и библиотеки. Большая часть программ и библиотек, не требующихся для загрузки или восстановления системы, хранятся в этом каталоге.
- /var — каталог содержит часто изменяемые файлы: системные журналы (/var/log), конфигурационные файлы, сообщения электронной почты, веб-сайты, файловые архивы ftp и другие данные.

Подробное описание структуры каталогов системы можно получить, используя следующую команду:

```
[user@station ~]$  man 7 hier
```

Путь к файлу, находящемуся в корневом каталоге, записывается так: **/file**. Путь к файлам в других каталогах, также начинается от «/» и содержит список всех каталогов на пути от корневого каталога до файла. Например, **/home/student/file**. Другой вариант записи пути на файл — начинать запись не от корня каталогов, а от текущего рабочего каталога. При этом текущий каталог обозначается символом «**.**», а родительский каталог - «**..**». Например, если вы находитесь в каталоге **/home/student/dir1/**, а хотите посмотреть файл

/home/student/dir2/file, то относительный путь до него будет **../dir2/file.** Еще один вариант записи пути - путь относительно домашнего каталога. Путь до вашего домашнего каталога можно заменить на символ «~». Например, путь к файлу /home/student/dir2/file можно записать так: ~/dir2/file.

Таким образом, к файлу всегда можно обратиться тремя способами. Путь к файлу относительно корневого каталога всегда начинается с «/» и называется **абсолютным**. Путь к файлу относительно текущего каталога записывается без «/» и называется **относительным.** Запись пути относительно домашнего каталога обычно используется при обращении к файлам, находящимся в домашнем каталоге и его подкаталогах.

Создание каталогов:

Для создания каталогов используется команда **mkdir** (mkdir — сокращенно от «make directory»). В качестве аргумента команде передается имя каталога, который требуется создать.

Для создания каталога с подкаталогами необходимо использовать ключ -p, так как иначе будет выведена ошибка, и команда не отработает.

Пример использования команды **mkdir**:

```
[user@station ~]$ mkdir -p dir1/dir2
```

Просмотр содержимого каталогов:

Для просмотра содержимого каталогов используется команда **ls** (ls — сокращенная форма глагола «list»). Команда ls без аргументов отображает содержимое текущего рабочего каталога. В качестве аргумента можно указать ссылку на каталог, содержимое которого хочется просмотреть. Наиболее часто используемые ключи команды **ls**:

-l — выводит «длинный список» каталогов и файлов, указывая для каждого элемента его тип (каталог или файл), права доступа, владельца, размер и другие данные.

-a — выводит полный список каталогов и файлов, включая скрытые файлы (их названия начинаются с символа «.»)

-R используется для рекурсивного вывода содержимого каталога. При этом выводится не только содержимое каталога, указанного в качестве аргумента команды, но и содержимое всех подкаталогов.

Пример использование команды **ls**:

```
[user@station ~]$ ls
dir1 file.txt
[user@station ~]$ ls -la
```

```
drwx------     31   user user 4096 Jun  19   00:32     .
drwxr-xr-x     6    root root 4096 Jul  12   23:19     ..
drwxr-xr-x     3    user user 4096 Aug  1    04:25     dir1
-rwxr-xr-x     1    user user 2252 Jul  30   20:07     file.txt
```

В результате вывода команды ls -l показывается «длинный список». В него входит:

- - тип файла (d — каталог, - — простой файл, l — символьная ссылка, c — символьное устройство, b — блочное устройство, s — сокет, p — канал);
- права доступа к файлу (будет рассмотрено в следующей работе);
- количество ссылок на файл;
- имя владельца;
- имя группы владельца;
- размер файла (в байтах);
- временной штамп;
- имя файла.

Просмотр файлов:

Для просмотра файлов существует несколько команд, простейшей из которых является **cat**. Если передать ей список файлов в качестве аргумента, то их содержимое будет «склеено» и выведено на экран. Если указать только один файл, то выведется содержимое этого файла.

Пример использования команды **cat**:

```
[user@station ~]$ cat /etc/hosts
```

Перенаправление потока вывода в файл:

При выполнении команд **ls** и **cat** результат их работы отображается на экране. В UNIX большинство команд, которые выводят текст на экран, используют понятие **стандартный поток вывода**. По умолчанию он связан с терминалом. Оболочка bash позволяет перенаправлять стандартный поток вывода в другие места. Например, в файл. Для этого используется символ «>».

Пример использования:

```
[user@station ~]$ ls /tmp > file.txt
```

Если файл file.txt существует, то его содержимое будет перезаписано выводом команды ls. Если этого файла не существует, то он будет создан.

Для добавления данных в файл без затирания уже записанной в него ранее информации используются символы «>>». При этом новые данные будут добавлены в конец файла.

При написании Bash-скриптов часто используется перенаправление потока стандартного вывода или потока ошибок, возникших в ходе выполнения скрипта, в специальное устройство **/dev/null** (пустое устройство). Запись в него происходит успешно независимо от объема переданной информации. Чтение из **/dev/null** эквивалентно считыванию конца файла (EOF). Например, для перенаправления потока стандартного вывода при выводе на экран содержимого файла **file1.txt** в **/dev/null** используется следующая команда:

```
[user@station ~]$ cat file1.txt > /dev/null
```

При выполнении этой команды содержимое файла не будет выведено на экран.

Для перенаправления ошибок в **/dev/null** при попытке просмотра содержимого домашнего каталога пользователя root, на просмотр которого у вас нет прав, используется следующая команда:

```
[user@station ~]$ ls /root/ 2> /dev/null
```

Создание файлов:

Пустые файлы можно создать несколькими способами.

Например, команда touch используется для обновления данных о временах модификации и последнего доступа к файлу. При отсутствии файла, к которому обращается команда, будет создан пустой файл.

Аналогично перенаправлению потока вывода в файл можно создавать новые пустые файлы, перенаправляя в файл отсутствие данных.

Пример создания пустого файла:

```
[user@station ~]$ > file.txt
```

Копирование файлов:

С помощью команды **cp** (copy) можно создавать копии файлов и каталогов.

Примеры использования команды **cp**:

`[user@station ~]$ cp source target` — файл с именем **source** копируется в файл с именем **target**

`[user@station ~]$ cp source dir/` — файл с именем **source** копируется в каталог **dir** с тем же именем

`[user@station ~]$ cp -r source target` — каталог с именем **source** копируется в каталог с именем **target**

Перемещение файлов:

С помощью команды mv (move) файлы можно перемещать из одного каталога в другой или менять имя файла.

Примеры использования команды **mv**:

[user@station ~]$ mv source target — файл **source** переименовывается в файл **target**

[user@station ~]$ mv source dir/ —файл **source** перемещается в каталог **dir/**

Команда mv интересна тем, что принцип ее работы тесно связан с файловой системой: UNIX воспринимает имя файла как нечто внешнее по отношению к его содержимому. Несмотря на то, что название команды происходит от слова «перемещение», она редко занимается перемещением данных. Вместо этого файловая система просто изменяет имя. Если имя файла изменяется с **/dir/file** на **/dir/newfile**, то это называется переименованием. Если имя файла изменяется с **/dir/file** на **/newdir/file**, то это называется перемещением. Если имя файла изменяется с **/dir/file** на **/newdir/newfile**, то это — перемещение с переименованием. Но по сути в UNIX все это является одним и тем же: изменением полного имени файла.

Удаление файлов и каталогов:

С помощью команды **rm** (remove) файлы можно удалять.

Для удаления каталога используется команда rmdir. Для ее использования необходимо предварительно удалить все файлы и подкаталоги, так как rmdir удаляет только пустые каталоги.

Для рекурсивного удаления каталога с файлами и подкаталогами можно использовать команду rm с ключом -r.

Пример использования команды **rm**:

[user@station ~]$ rm file — удаление файла

[user@station ~]$ rm -r dir1 — удаление каталога и всех вложенных файлов

Ссылки

Файл в UNIX состоит из трех частей:

- inode (индексный дескриптор, в котором хранится вся метаинформация о файле или каталоге кроме непосредственно данных и имени объекта). Эту информацию можно увидеть, выполнив команду ***stat имя_файла***.

- dentry (каталожная запись). Эту информацию можно увидеть, выполнив команду ***ls -i имя_файла***.
- непосредственно содержимое объекта, хранимое на файловой системе. Эту информацию можно увидеть, выполнив команду ***cat имя_файла***.

Существуют два типа ссылок: жесткие и символьные. **Жесткие ссылки** привязывают многочисленные dentries к одному inode. **Символьные ссылки** — это особые inode, которые указывают на другие имена файлов.

Для создания ссылок используется команда **ln.**

Пример использования жестких ссылок:

Иногда требуется, чтобы один и тот же файл находился в нескольких местах или имел два разных имени. Для этого используются ссылки на файлы. Предположим, пользователи user1 и user2 хотят вместе работать над одним файлом (file.txt) и иметь возможность пользоваться работой друг друга. Вместо того, чтобы копировать и обновлять файл каждый раз, когда другой пользователь изменил что-то, и постоянно синхронизировать свои личные копии, они решают создать жесткую ссылку.

user1 создал каталог ~/dir, выдав другим пользователям права на запись в этот каталог (о правах доступа в следующей работе). user2 делает то же самое в своем домашнем каталоге. Затем user1 создает файл ~/dir/file.txt и использует команду ln, чтобы создать ссылку на файл в каталоге dir пользователя user2.

```
[user1@station ~]$ ls -ld dir/
drwxrwxr-x     2    user1     group     4096 Jul 13 05:45 dir/
[user1@station ~]$ touch dir/file.txt
```

[user1@station ~]$ chgrp group dir/file.txt – выдача прав доступа к файлу группе group, в которую входит пользователь user2

```
[user1@station ~]$ ln dir/file.txt /home/user2/dir/file.txt
```

Поскольку была создана ссылка на файл, а не его копия, то по сути мы работаем с одними и теми же данными, обращаясь к ним по двум разным именам. Когда пользователь user2 редактирует файл /home/user2/dir/file.txt, он также редактирует файл /home/user1/dir/file.txt.

Если один из пользователей удалит файл ~/dir/file.txt, то из системы удалится только имя этого файла. Файл по-прежнему будет виден и доступен второму пользователю по второму имени в его домашнем каталоге. Сам файл будет удален из файловой системы только тогда, когда будут удалены все его имена.

Жесткие ссылки нельзя использовать в следующих случаях:

- Нельзя создавать жесткие ссылки на каталог.

- Жесткая ссылка не может пересекать границы файловой системы.

Пример использования символьных ссылок:

Пользователь user держит в своем домашнем каталоге семь файлов со списками дел на каждый день недели.

```
[user@station ~]$ ls
friday.todo    monday.todo    saturday.todo  sunday.todo
     thursday.todo  tuesday.todo   wednesday.todo
```

Чтобы не забывать, какой именно сегодня день недели, и в каком файле нужно смотреть список дел на сегодня, ему лучше создать файл today.todo, который он будет обновлять каждое утро. Для этого он решает использовать символьную ссылку. Поскольку сегодня вторник, он использует ту же команду ln, которая использовалась для создания жесткой ссылки, но добавляет к ней ключ -s, чтобы указать правильный тип ссылки.

```
[user@station ~]$ ls
friday.todo    monday.todo    saturday.todo  sunday.todo
     thursday.todo  tuesday.todo   wednesday.todo
[user@station ~]$ ln -s tuesday.todo today.todo
[user@station ~]$ ls -l
total 32
-rw-rw-r-- 1 user group 138   Jul 14 09:54   friday.todo
-rw-rw-r-- 1 user group 29    Jul 14 09:54   monday.todo
-rw-rw-r-- 1 user group 578   Jul 14 09:54   saturday.todo
-rw-rw-r-- 1 user group 252   Jul 14 09:54   sunday.todo
-rw-rw-r-- 1 user group 519   Jul 14 09:54   thursday.todo
lrwxrwxrwx 1 user group 12    Jul 14 09:55   today → tuesday.todo
-rw-rw-r-- 1 user group 37    Jul 14 09:54   tuesday.todo
-rw-rw-r-- 1 user group 657   Jul 14 09:55   wednesday.todo
```

Созданная ссылка имеет тип l (символьная ссылка), что можно увидеть в первом столбце вывода команды ls -l. Также в выводе указано, на какой файл ссылается эта ссылка.

В отличие от жесткой ссылки, символьная ссылка правда создает новый файл со своим inode. Однако этот файл имеет тип «символьная ссылка» и вместо реальных данных содержит имя другого файла.

Если пользователь удалит или переименует исходный файл, на которой сделана символьная ссылка, то ссылка становится сломанной, то есть она отсылает пользователя к несуществующему файлу.

Поиск файлов и содержимого в них

Команда find используется для поиска в файловой системе файлов, которые соответствуют особым критериям. Почти все параметры файла могут быть указаны, например, его имя, размер, время последнего изменения, даже число ссылок. Единственное ограничение — find не позволяет искать файлы по их содержимому.

Синтаксис использования команды **find**:

```
find [каталог поиска] [критерии поиска] [действие]
```

Для поиска файла в корне каталогов по имени и вывода списка файлов без выполнения каких-либо действий может использоваться следующая команда:

```
[user@station ~]$ find / -name "*.conf"
```

При использовании **find** есть возможность указать действия, которые необходимо выполнить при нахождении файла с именем, удовлетворяющим выражению поиска. Для этого используется параметр **exec.**

Пример использования команды **find** для удаления всех файлов, размер которых превышает 100 Мб:

```
[user@station ~]$ find / -size +100M -exec /bin/rm {} \
```

Для более близкого знакомства с командой **find** воспользуйтесь справкой, вызвав ее командой **man find** на своей рабочей станции.

Команда **grep** традиционно используется для выборки из всех данных только тех, что нас интересуют.

Пример использования команды **grep**:

```
[user@station ~]$ grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

Первый аргумент команды — текст для поиска, остальные аргументы — файлы для поиска. Если команда вызывается только с одним аргументом, то в качестве источника данных она использует стандартный ввод.

Подстановка имен файлов (Pathname Expansion)

Для упрощения работы в оболочке bash можно использовать подстановки, обращаясь к файлам и каталогам.

«~» (Tilde Expansion) обозначает домашний каталог пользователя. Например, команда *cd ~* заменяет команду *cd /student/group/username* и перемещает вас в домашний каталог. Для

перехода в домашний каталог другого пользователя укажите его имя после символа «~», например «~root».

Пример:

```
[user1@station /]$ cd ~
[user1@station ~]$ cd ~user2
[user1@station /home/user2]$
```

Символ «*» заменяет любое число произвольных символов в имени файла или каталога.

Пример:

```
[user@station ~]$ ls
dir1 f3 file file1 file2 q1
 [user@station ~]$ ls file*
file file1 file2
[user@station ~]$ ls *1
dir1 file1 q1
```

Символ «?» аналогичен по своим свойствам символу «*», но заменяет один произвольный символ.

Пример:

```
[user@station ~]$ ls
dir1 f3 file file1 file2 q1
 [user@station ~]$ ls file?
file1 file2
[user@station ~]$ ls ?1
q1
```

Использование квадратных скобок ([]) позволяет выбирать произвольные символы для подстановки.

Пример:

```
[user@station ~]$ ls
dir1 f3 file file1 file2 q1
[user@station ~]$ ls [f,q]*1
file1 q1
```

Каналы

Поток вывода (stdout) из одного процесса может быть связан с потоком ввода (stdin)

другого процесса. Это носит название «канал». Для создания канала между двумя командами в bash используется символ «|».

Пример использования каналов:

Например, пользователю нужно найти самые большие файлы в каталоге /etc. Сначала он составляет команду find, которая найдет все файлы, размер которых больше 100Кб.

```
[user@station ~]$ find /etc -size +100k
```

Заметив, что команда выводит список файлов без сортировки, он решает отсортировать их в алфавитном порядке. Можно перенаправить вывод команды find в файл, а затем отсортировать список, используя команду sort. Вместо этого он решает создать канал, направив результат поиска на вход команды sort.

```
[user@station ~]$ find /etc -size +100k | sort
```

При этом файлы будут отсортированы в алфавитном порядке.

Задание 1. Знакомство с командами для работы с файлами и каталогами.

1. Определите полное имя вашего домашнего каталога, объясните структуру абсолютного пути к каталогу.
2. Выведите на экран содержимое корневого каталога системы. Опишите назначение основных каталогов системы.
3. Выведите на экран информацию о вашем пользователе в файле **/etc/passwd**, используя команду grep.
4. Выведите «длинный список» файлов (в том числе скрытых), содержащихся в вашем домашнем каталоге. Опишите, что обозначают все столбцы списка.
5. Изучите самостоятельно для каких целей служат спецсимволы ^, ! и фигурные скобки ({}), создайте файлы, которые можно использовать для проверки работы этих спецсимволов.

Задание 2. Создание структуры веб-сайта

1. Создайте каталоги **~/html** и **~/archive**.
2. Выведите на экран содержимое каталога /usr/share. Ограничьте длину результата 5 строками. Запишите результат работы команды в файл **index.html** в каталоге с сайтом.
3. Проверьте количество строк в файле **index.html**. Выведите на экран содержимое файла **index.html**.
4. Переименуйте каталог **html** в **html_public**.
5. Создайте копию файла **index.html** в **~/archive**.
6. Выведите список файлов в домашнем каталоге, отсортировав их в порядке, обратном алфавитному. Сохраните вывод в файл **~/html_public/home.html**.
7. Скопируйте в домашний каталог файл **home.html**.

8. Создайте символьную ссылку к файлу **index.html** с именем **~/html_public/link_s.html** и жесткую ссылку к этому же файлу с именем **~/html_public/link_h.html**.
9. Удалите файл **index.html** так, чтобы ссылка **link_s.html** оказалась «сломанной». Попробуйте открыть содержимое файлов **link_s.html** и **link_h.html**. Прокомментируйте результат.
10. Определите место, занимаемое в системе вашим сайтом (каталогами **html_public** и **archive**).

Контрольные вопросы:

1. Какие основные каталоги системы вы знаете? Каково их назначение?
2. Как обратиться к файлу, который находится в каталоге, расположенном выше относительно текущего в дереве каталогов системы?
3. Какие условия поиска файлов вы знаете? Как их можно комбинировать?
4. Как узнать имя владельца файла и размер файла?
5. Какую информацию содержит пустой каталог?
6. Как осуществить поиск файлов в системе каталогов по фрагментам текста файлов?
7. Как осуществить поиск файлов по их типу и владельцу?
8. Назовите известные вам способы создания пустых файлов.
9. Какие типы файлов в системе UNIX вы знаете?
10. Сколько ссылок можно создать на один файл из разных каталогов?
11. В чем разница между жесткой и символьной ссылкой? Что такое «сломанная символьная ссылка»?
12. Какими возможностями обладает команда sort?

Лабораторная работа №3. Разграничение прав доступа в ОС UNIX

Управление правами доступа к файлам и каталогам

Основные понятия:

- права доступа к объекту (файлу или каталогу)
- владелец объекта (файла или каталога)
- группа-владелец объекта (файла или каталога)
- права на чтение, запись и исполнение объекта (файла или каталога)

Используемые команды:

chmod	изменение прав доступа к объекту
chown	изменение владельца объекта
chgrp	изменение группы-владельца объекта
umask	изменение шаблона прав доступа для новых файлов и каталогов
getfacl	просмотр списка контроля доступа
setfacl	изменение списка контроля доступа

Механизм разграничения прав доступа позволяет ограничивать доступ пользователей к объектам системы – каталогам и файлам.

Права доступа подразделяются на три типа: чтение (read), запись (write) и выполнение (execute). Эти типы прав доступа могут быть предоставлены трем классам пользователей: пользователю-владельцу файла (u), выбранной группе пользователей (g) и всем остальным пользователям (o). Владелец файла обычно является членом группы, пользователям которой выдаются права доступа.

Разрешение на чтение позволяет пользователю читать содержимое файлов, а в случае каталогов - просматривать перечень имен файлов в каталоге (используя, например, **ls**). Разрешение на запись позволяет пользователю писать в файл и изменять его. Для каталогов это дает право создавать новые файлы и подкаталоги, или удалять файлы в этом каталоге. Разрешение на выполнение позволяет пользователю выполнять файлы (как бинарные программы, так и командные файлы). Разрешение на выполнение применительно к каталогам означает возможность переходить к каталогу, используя, например, команду cd.

Права доступа к обычным файлам и каталогам:

	(r) чтение	**(w) запись**	**(x) выполнение**
Обычный файл	просмотреть содержимое файла	изменить файл	использовать файл как команду

Каталог	вывести список содержащихся в каталоге подкаталогов и файлов	добавить или удалить файлы	работа с известным файлом внутри каталога

Для просмотра прав доступа к файлу или каталогу можно воспользоваться командой **ls -l**.

Пользователь, создающий объект, становится его владельцем, а основная группа, в которую входит пользователь, становится группой, имеющий доступ с указанными правами к объекту. Пользователь, создавший объект, не может изменить владельца, но может изменить группу и права доступа всех категорий пользователей к объекту. Изменить владельца объекта может только суперпользователь root.

Для изменения владельца файла или каталога используется команда **chown**. Синтаксис команды:

chown [-ключи] новый пользователь:[новая группа] имя файла

Для рекурсивного изменения владельца каталога и вложенных файлов используется ключ **-R**. Если группу-владельца изменять не требуется, то эта часть команды опускается.

Для изменения группы-владельца файла или каталога воспользуйтесь командой **chgrp.** Синтаксис команды:

chgrp [-ключи] новая_группа имя_файла

Для изменения владельца каталога и всех вложенных файлов используется ключ **-R.**

Существует два формата записи прав доступа: символический, который мы видим, работая с командой **ls -l** (rwxrwxrwx или r-x------) и восьмеричный (777 или 002). Как работать с форматами записи прав доступа?

Файл имеет три разных типа разрешений к доступу чтение (r), запись (w) и выполнение (x) для трех классов пользователей: пользователь (u), группа (g) и остальные (o). При восьмеричной записи каждый тип пользователя с правами доступа получает разряд: место "сотен" - для пользователя (u), место "десяток" для группы (g), а место "единиц" - для остальных (o). Каждый тип доступа получает цифровое обозначение: чтение (r) получает 4 (100), запись (w) получает 2 (010), а выполнение (x) получает 1 (001). Символы в восьмеричной записи - это совокупность прав доступа для данного класса пользователя с правами доступа.

755 = rwxr-xr-x

7 = 4 + 2 + 1 = rwx для пользователя (u)
5 = 4 + 0 + 1 = r-x для группы (g)
5 = 4 + 0 + 1 = r-x для остальных (o)

640 = rw-r-----

6 = 4 + 2 + 0 = rw- для пользователя (u)ser
4 = 4 + 0 + 0 = r– для группы (g)
0 = 0 + 0 + 0 = --- для остальных (o)

Используя команду **chmod** можно изменять права доступа к файлу. Синтаксис команды:

chmod [-ключи] права_доступа имя_файла_или_каталога

Для рекурсивного изменения прав доступа используется ключ **-R**, изменяющий права для каталога и всех файлов внутри него.

Примеры использования команды **chmod**:

Исходный файл:

```
-rw-rw-r--     1    user user 42    Jan 16 08:09  file.txt
```

Лишить группу права записи:

```
[user@station ~]$ chmod g-w file.txt
-rw-r—r-- 1    user user 42    Jan 16 08:09  file.txt
```

Наделить пользователя-владельца и группу правом исполнения:

```
[user@station ~]$ chmod ug+x file.txt
-rwxrwxr--     1    user user 42    Jan 16 08:09  file.txt
```

Наделить всех других пользователей правом записи:

```
[user@station ~]$ chmod o+w file.txt
-rw-rw-rw-     1    user user 42    Jan 16 08:09  file.txt
```

Лишить группу и всех других пользователей всех прав:

```
[user@station ~]$ chmod go-rwx file.txt
-rw-------     1    user user 42    Jan 16 08:09  file.txt
```

Лишить всех права записи:

```
[user@station ~]$ chmod a-w file.txt
-r--r--r--     1    user user 42    Jan 16 08:09  file.txt
```

Лишить пользователя-владельца и всех других пользователей права чтения:

```
[user@station ~]$ chmod uo-r file.txt
--w-rw----     1    user user 42    Jan 16 08:09  file.txt
```

Наделить группу и других пользователей правами чтения и исполнения, но не записи

```
[user@station ~]$ chmod go=rx file.txt
-rw-r-xr-x     1    user user 42    Jan 16 08:09  file.txt
```

Наделить пользователя-владельца правами исполнения и лишить группу права записи:

```
[user@station ~]$ chmod 744 file.txt
-rwxr--r--     1    user user 42    Jan 16 08:09  file.txt
```

Наделить всех пользователей всеми правами:

```
[user@station ~]$ chmod 777 file.txt
-rwxrwxrwx     1    user user 42    Jan 16 08:09  file.txt
```

Лишить других пользователей права чтения:

```
[user@station ~]$ chmod 660 file.txt
-rw-rw-r--     1    user user 42    Jan 16 08:09  file.txt
[user@station ~]$ chmod 744 file.txt
-rw-rw-r--     1    user user 42    Jan 16 08:09  file.txt
```

При выставлении прав доступа для создаваемого файла, система считает права доступа, исходя из режимного глобального кода по умолчанию 666 (rw-rw-rw-) для файлов и 777 (rwxrwxrwx) для каталогов. Затем система применяет umask процесса, который создал файл (например, bash). Любые значения, установленные в umask, складываются с глобальным кодом. Например, umask со значением 002 установит права доступа к файлу по умолчанию 664:

Стандартный код системы: *666 → rw- rw- rw-*

umask: *002 → --- --- -w-*

Стандартные права: *664 → rw- rw- r--*

Если umask установлен в 135, то будет создан файл с правами 643:

Стандартный код системы: *666 → rw- rw- rw-*

umask: *135 → --x -wx r-x*

Права на файл: *643 → rw- r-- -w-*

Для изменения umask оболочки bash используется команда **umask**. Когда команда выполняется без параметров, она сообщает о текущем значении umask. Когда она выполняется с восьмеричной записью в качестве единственного аргумента, текущее значение umask изменяется на новое.

Пример использования команды **umask**:

```
[user@station ~]$ umask
```

```
0002
[user@station ~]$ umask 077
0077
```

Задание 1.

1. Создайте в своем домашнем каталоге подкаталог **labs**.
2. Запустите следующую команду, чтобы создать группу файлов, которыми необходимо распорядиться:
   ```
   [user@station ~]$ touch labs/lab{1,2,3,4}.{draft,final}.{txt,pdf}
   ```

Если вы правильно ввели команду, у вас появится следующие файлы:

```
[user@station ~]$ ls labs/
lab1.draft.pdf lab2.draft.pdf lab3.draft.pdf lab4.draft.pdf
lab1.draft.txt lab2.draft.txt lab3.draft.txt lab4.draft.txt
lab1.final.pdf lab2.final.pdf lab3.final.pdf lab4.final.pdf
lab1.final.txt lab2.final.txt lab3.final.txt lab4.final.txt
```

3. Задайте права доступа к вашему домашнему каталогу так, чтобы другие пользователи могли получить доступ к подкаталогу **labs**, но не могли просматривать список файлов в домашнем каталоге.
4. Убедитесь, что другие пользователи могут получить доступ и посмотреть список файлов в каталоге **labs**. Для этого попросите другого пользователя попробовать открыть ваш каталог.
5. Члены вашей группы должны будут помочь вам с выполнением лабораторных работ. Создайте каталог **~/labs/draft** и выдайте вашей группе права доступа к этому каталогу так, чтобы любой мог просмотреть список содержимого каталога, но только студенты вашей группы могли создать новые файлы. Переместите все обычные файлы, которые содержат слово «draft» в своем названии, в ваш вновь созданный каталог и замените группу-владельца файла на вашу группу.
6. За итоговый вариант файлов с выполненными лабораторными работами отвечаете вы. Создайте каталог **~/labs/final**. Вы не хотите, чтобы кто-то мог изменять вашу работу, поэтому установите такие права доступа, чтобы только вы имели доступ к каталогу на запись. Вы должны иметь полный доступ к этому каталогу, а все другие студенты вашей группы должны иметь возможность просматривать файлы этого каталога. Переместите все обычные файлы, которые содержат в своем названии слово «final», во вновь созданный каталог.
7. Работа над первой лабораторной работой закончилась, поэтому позаботьтесь о том, чтобы вы случайно не удалили файл **lab1.final.pdf** и **lab1.final.txt** из каталога **final**.
8. Вы хотите иметь место, где можно записывать свои планы по выполнению будущих работ, вы не хотите, чтобы кто-либо увидел ваши мысли. Создайте каталог **~/labs/planning**, и установите такие права доступа, чтобы вы имели полный доступ, а все остальные не имели доступа вообще.
9. Вы заинтересованы в том, чтобы получать от любых людей подсказки по выполнению работ, но вы хотели бы, чтобы этот процесс был анонимным. Создайте каталог **~/labs/submissions** и установите такие прав доступа, чтобы любой человек мог создавать новые файлы, но только вы могли видеть список содержимого каталога.

Проверьте, что все работает, попросив соседа создать новый файл в вашем каталоге с именем **~/labs/submissions/suggestion.txt**.

10. На всякий случай скопируйте команду **/bin/ls** как файл **~/labs/ls**. Поскольку вы не хотите, чтобы кто-то запустил этот исполняемый файл, удалите все права доступа на выполнение.
11. Поскольку другие люди будут пользоваться этим каталогом, вы слегка обеспокоены по поводу новых файлов, которые вы будете создавать. Добавьте подходящую команду в конец вашего файла **~/.bashrc**, так чтобы **umask** вашей оболочки автоматические был установлен на 077 при запуске.

Списки контроля доступа

Основные понятия:

- запись списка контроля доступа

Используемые команды:

getfacl	Просмотр списка контроля доступа для файла или каталога
setfacl	Установка списка контроля доступа для файла или каталога

Списки контроля доступа используются для выдачи доступа к файлу дополнительным группам и пользователям, если функционала стандартного разграничения прав доступа к файлу оказывается недостаточно.

Команда **getfacl** используется для просмотра списка контроля доступа для файла. Если списки контроля доступа не заданы, но команда выведет на экран стандартные права доступа UNIX для выбранного файла.

```
[user1@station ~]$ getfacl file1.txt
# file: file1.txt
# owner: user1
# group: users
user:: rw-
group::rw-
other::---
```

С помощью команды setfacl можно добавить списки контроля доступа для файла. Например, чтобы разрешить членам группы **developers** читать файл **file1.txt**, пользователю **user2** — читать и обновлять файл, а пользователю **user3** — только читать файл, можно воспользоваться следующими командами:

```
[user1@station ~]$ setfacl -m g:developers:rw file1.txt
[user1@station ~]$ setfacl -m u:user2:rw file1.txt
[user1@station ~]$ setfacl -m u:user3:r file1.txt
```

После выполнения этих действий команда **getfacl** выведет следующую информацию о файле:

```
[user1@station ~]$ getfacl file1.txt
```

```
# file: file1.txt
# owner: user1
# group: users
user::rw-
user:user3:r--
user:user2:rw-
group::rw-
group:developers:rw-
mask::rw-
other::---
```

Каждая строка, выводимая командой **getfacl**, называется записью, и может быть одного из следующих типов:

Запись	Имя acl	Синтаксис	#	Комментарий
пользователь-владелец файла	ACL_USER_OBJ	user::	1	стандартные права доступа для пользователя-владельца файла
отдельные пользователи	ACL_USER	user:	0+	права доступа для дополнительных пользователей
группа-владелец файла	ACL_GROUP_OBJ	group::	1	стандартные права доступа для группа-владельцев файла
отдельные группы	ACL_GROUP	group:	0+	права доступа для дополнительных групп
маска	ACL_MASK	mask::	0 или 1	максимально возможные права доступа, разрешенные для дополнительных пользователей и любой группы
все остальные	ACL_OTHER	other::	1	права доступа для любого пользователя, который не входит ни в одну из вышеназванных категорий

Все записи кроме маски должны быть заданы.

Когда процесс пытается получить доступ к файлу, выполняются следующие действия:

1. Если пользователь процесса — владелец файла, то используются стандартные права пользователя-владельца файла.
2. Если права пользователя процесса заданы одной из записей, описывающих права доступа дополнительных пользователей, то применяются соответствующие права.
3. Если группы процесса содержат группу-владельца файла, но маски нет, то применяются стандартные права доступа группы.
4. Если список групп процесса содержит любую группу, права которой заданы одной из записей, описывающих права доступа дополнительных групп, то применяются соответствующие права.
5. В остальных случаях применяются права доступа для других пользователей.

Для управления списками контроля доступа используется команда **setfacl**, которая

позволяет изменить (или добавить) записи (ключ -m) или удалить записи (ключ -x). Можно указать несколько записей, разделенных запятыми, соблюдая следующий синтаксис:

setfacl [ключ] {user|group|other|mask}:[username|groupname]:{r|-}{w|-}{x|-} файл

Первое поле уточняет тип записи и может быть сокращено до первой буквы (u|g|o|m). Второе поле связано только с пользователем и группой и задает пользователя или группу. Если оно пустое, используются пользователь-владелец или группа-владелец файла. В последнем поле указываются права доступа.

Пример использования команды **setfacl**:

```
[user1@station ~]$ setfacl -m g:physics:rw,u:petr:rw,u:ivan:r file1.txt
[user1@station ~]$ getfacl file1.txt
# file: file1.txt
# owner: user1
# group: users
user::rw-
user:ivan:r--
user:petr:rw-
group::rw-
group:physics:rw-
mask::rw-
other::---
[user1@station ~]$ setfacl -x u:ivan file1.txt
[user1@station ~]$ getfacl file1.txt
# file: file1.txt
# owner: user1
# group: users
user::rw-
user:petr:rw-
group::rw-
group:physics:rw-
mask::rw-
other::---
```

У каталогов в отличие от файлов есть два набора списков контроля доступа: стандартные и по умолчанию. Стандартные списки отвечают за доступ к самому каталогу, точно так, как указано выше. Списки по умолчанию не применяются к каталогу, а вместо

этого устанавливают списки контроля доступа для каждого файла, созданного внутри этого каталога.

Списки контроля доступа по умолчанию задаются так же, как и стандартные, но с добавлением в начало четвертого поля слова default (которое можно сократить до "d").

[user1@station ~]$ chmod g+s dir – используется для того, чтобы файлы в каталоге по умолчанию имели ту же группу-владельца, что и каталог

```
[user1@station  ~]$ setfacl -m g:physics:r-x dir/
[user1@station  ~]$ setfacl -m default:g:physics:r dir/
[user1@station ~]$ getfacl dir/
# file: dir
# owner: user1
# group: users
user::rwx
group::rwx
group:physics:r-x
mask::rwx
other::---
default:user::rwx
default:group::rwx
default:group:physics:r--
default:mask::rwx
default:other::---
```

Задание 2.

1. Создайте подкаталог **shared** в вашем домашнем каталоге. Узнайте имя пользователя вашего одногруппника, сидящего за соседним компьютером.
2. Создайте файл **doc1.txt** в новом каталоге. Проверьте стандартные права доступа к файлу. Используя списки контроля доступа, дайте вашему соседу права на чтение созданного файла.
3. Проверьте, что сосед может получить доступ к созданному файлу. В чем может быть причина, если ему не удалось открыть файл?
4. Договоритесь с соседним пользователем о создании общего каталога. Настройте списки контроля доступа по умолчанию для каталога **shared** так, чтобы ваш сосед мог создавать новые файлы в нем и иметь права на их чтение и запись.
5. Попросите соседа создать файл **file1.txt** и записать в него строку «**This text is created by user1**». Откройте созданный им файл и допишите в него вторую строку «**This text is added by user2**».

Контрольные вопросы:

1. Как кодируются в атрибутах файла и каталога права доступа? Какие форматы записи прав бывают?
2. Кто может изменять права доступа к файлам?
3. Какие команды для изменения символьных кодов прав доступа Вы знаете? Перечислите и расскажите о назначении каждой из команд.
4. В чем разница в применении команд chmod и umask?
5. Какие команды обработки файлов разрешают (или запрещают) права на чтение, запись и выполнение?
6. Какие команды обработки каталогов разрешают (или запрещают) эти же права?
7. Что означает право на выполнение, применительно к каталогу?
8. Какими правами надо обладать, чтобы удалить файл или каталог?
9. В чем разница между обычными списками контроля доступа и списками контроля доступа по умолчанию?

Лабораторная работа №4. Управление процессами

Основные понятия:

- процесс
- задание
- интерактивный режим
- фоновый режим
- приоритет

Используемые команды:

bg	перевести задание в фоновый режим
fg	перевести задание в интерактивный режим
jobs	вывести список заданий текущего shell
kill	послать процессу или заданию сигнал
killall	послать процессу сигнал, используя его имя
nice	запустить команду с измененным приоритетом
nohup	запустить команду с защитой от прерывания
pkill	послать сигнал, процессам удовлетворяющим некоторым критериям
ps	вывести список процессов
renice	изменить приоритет процесса

Базовым понятием в ОС UNIX является процесс. Процесс в первом приближении можно определить как экземпляр выполняющейся программы с необходимым для ее выполнения набором ресурсов. В рамках ОС процесс представлен некоторым набором структур данных. Базовая структура данных – таблица процессов, в которой каждый процесс представлен одной записью. Поля этой структуры описывают различные характеристики процесса (перечислены некоторые из них):

PID – уникальный числовой идентификатор процесса;

PPID – идентификатор родительского процесса;

STATUS – состояние процесса;

TIME – время выполнения процесса (суммарно в режиме ядра и режиме задачи);

START – время запуска процесса;

TTY – терминал процесса;

PRI – текущий приоритет процесса;

NI – значение nice;

RUID – реальный идентификатор владельца процесса;

EUID – эффективный идентификатор владельца процесса;

RGID – реальный идентификатор группы;

EGID – эффективный идентификатор группы;

CPU – процент использования процессорного времени;

MEM – использование физической памяти.

Процесс в системе может находиться в некотором состоянии (поле STATUS). Только один процесс в каждый момент времени может выполняться на процессоре (иметь состояние R). Для многопроцессорных систем таких процессов может быть несколько. В рамках процесса может выполняться прикладная задача пользователя (пользовательский процесс), а может – системная задача (системный процесс). Особое значение имеет системный процесс init, который запускается в самом начале работы системы и является предком всех процессов. Его PID фиксирован и всегда равен 1. Этот процесс должен существовать всегда. Его завершение происходит только при останове или перезагрузке ОС.

Большинство командных интерпретаторов обеспечивает удобное управление процессами путем организации заданий. В данной лабораторной работе мы будем использовать командный интерпретатор bash. Все дальнейшее изложение материала будет вестись с учетом особенностей этого командного интерпретатора. Следует отметить, что другие командные интерпретаторы имеют аналогичные механизмы управления заданиями.

Задание – специальным образом оформленная совокупность команд. Задание может состоять из одного или нескольких процессов. Управление заданием осуществляется как единым целым. В заданиях, состоящих из нескольких процессов, посылка сигнала с помощью команды kill не заданию как таковому, а одному процессу нежелательно и может привести к непредсказуемым последствиям.

Задание обычно запускается в интерактивном режиме. Это означает, что терминал (клавиатура и дисплей) монопольно будут использоваться данным заданием. Ввод с клавиатуры будет передаваться процессам данного задания. Вывод будет направляться на экран. Управляющие комбинации клавиш будут приводить к посылке сигналов данному заданию.

Задание может быть запущено в фоновом режиме. В этом случае оно не может обращаться к терминалу. При необходимости ввод и вывод должны быть перенаправлены в соответствующие файлы. Управление таким заданием может осуществляться специальными командами командного интерпретатора bash.

Каждому заданию командный интерпретатор присваивает числовой идентификатор, который, в отличие от идентификатора процесса уникален только в пределах данного командного интерпретатора. Поэтому в командах, которые могут быть адресованы не только заданию, но и процессу (kill) перед идентификатором задания ставится символ %.

Задание

1. Выведите на экран листинг характеристик (в длинном и коротком форматах) процессов, инициализированных с Вашего терминала. Проанализируйте и объясните содержание каждого поля сообщения.
2. Выведите на экран листинг характеристик всех процессов. Используйте при необходимости конвейер с more для постраничного просмотра листинга. Какой процесс является родительским для большинства процессов? Что означает символ ? в поле управляющий терминал процесса?
3. Выведите на экран листинг процессов, запущенных конкретным пользователем. Какой ключ пришлось использовать? Что говорит значение ? в поле управляющий терминал процесса?
4. Разработайте и запустите простейшую процедуру в фоновом режиме с бесконечным циклом выполнения, предусматривающую, например, перенаправление вывода каких-то сообщений в файл или в фиктивный файл, и использующую команду sleep для сокращения частоты циклов процедуры.
5. Выполните п. 1. Объясните изменения в листинге характеристик процессов.
6. Понизьте значение приоритета процедуры. На что и как повлияет эта операция при управлении вычислительным процессом системы? Как отразятся ее результаты в описателях процессов?
7. Проанализируйте листинг процессов. Какой процесс является родительским для процедуры.
8. Выйдите из системы и войдите заново. Проанализируйте листинг процессов. Объясните изменения в системе.
9. Запустите процедуру в фоновом режиме, но предусмотрите ее защиту от прерывания при выходе из системы.
10. Выполните п.6. Объясните изменения PPID процедуры.
11. Завершите выполнение процесса процедуры.
12. Запустите процедуру в интерактивном режиме с перенаправлением вывода в соответствующий файл.
13. Переведите задание с процедурой в фоновый режим и проанализируйте сообщение на экране. Что пришлось дополнительно сделать? Как выглядят приостановленные процессы в листинге команды ps?
14. Переведите задание с процедурой в интерактивный режим и проанализируйте сообщение на экране.
15. Завершите выполнение процедуры и проанализируйте сообщение на экране.
16. Поставьте эксперимент, позволяющий определить, что будет происходить с процедурой, запущенной в фоновом режиме, в случае попытки ввода с клавиатуры. Как все-таки обеспечить ввод?
17. Поставьте эксперимент, иллюстрирующий относительные скорости выполнения нескольких фоновых процессов, запущенных с разными значениями поправки к приоритету. Завершите сразу все фоновые процессы одной командой pkill. Какие опции команды пришлось использовать для выделения фоновых процессов, запущенных с Вашего терминала?

Контрольные вопросы:

1. Какую комбинацию клавиш нужно использовать для принудительного завершения задания, запущенного в интерактивном режиме?
2. Какую комбинацию клавиш нужно использовать для приостанова задания, запущенного в интерактивном режиме?
3. Какая команда позволяет послать сигнал конкретному процессу?
4. Какая команда позволяет поменять поправку к приоритету уже запущенного процесса?
5. Какая команда позволяет запустить задание с пониженным приоритетом?
6. Какая команда позволяет запустить задание с защитой от прерывания при выходе из системы пользователя?
7. Какой процесс всегда присутствует в системе и является предком всех процессов?
8. Каким образом можно запустить задание в фоновом режиме?
9. Каким образом задание, запущенное в фоновом режиме, можно перевести в интерактивный режим?
10. Каким образом приостановленное задание можно перевести в интерактивный режим?
11. Что произойдет с заданием, выполняющимся в фоновом режиме, если оно попытается обратиться к терминалу?

Лабораторная работа №5. Программирование на языке shell.

Оболочка shell является основным средством взаимодействия пользователя с операционной системой в режиме командной строки. Shell обеспечивает интерпретацию и исполнение команд, вводимых пользователем из командной строки терминала, а также исполнение программ, написанных на одном из языков программирования и содержащихся (после обработки компилятором и редактором связей) в исполняемом бинарном файле. Такая программа может быть также написана на языке shell. В этом случае она называется shell-процедурой и не требует предварительной обработки компилятором и редактором связей перед ее исполнением, так как будет выполняться оболочкой в режиме интерпретации. Shell-процедура может содержать любые команды операционной системы (внешние команды), внутренние (встроенные в оболочку) команды, а также и вызовы других shell-процедур.

Исполнение shell-процедуры внешне ничем не отличается от исполнения любой команды Unix. Для передачи информации в процедуру существует 3 способа: через позиционные параметры при вызове процедуры, через глобальные переменные (переменные среды операционной системы), путем ввода информации командой read во время выполнения процедуры.

В процедуре можно использовать любые конструкции, допустимые в командной строке операционной системы (именованные и неименованные программные каналы, конвейеры команд, переадресацию ввода-вывода и т.д.).

Позиционные параметры процедуры в теле процедуры имеют имена: $0, $1, $2 и т.д., где $0 – имя процедуры, $1 – первый позиционный параметр процедуры, $2 – второй позиционный параметр процедуры и т.д.

Для организации ветвящихся алгоритмов обработки в процедурах используется конструкция

if <список команд 1> then <список команд 2> else <список команд 3> fi

или

if <список команд 1> then <список команд 2> fi

case <переменная> of образец 1) <список 1>;; образец 2) <список 2>;; ….. образец n) <список n> else <список команд> esac

В первой конструкции, если код возврата последней команды из списка 1 равен 0, то будет выполнен список команд 2; в противном случае будет выполнен список команд 3.

Во второй конструкции, если код возврата последней команды из списка 1 равен 0, то

будет выполнен список команд 2; в противном случае ничего не выполняется.

В третьей конструкции выполняется один из списков команд, для которого значение *переменной* совпадает с соответствующим образцом.

Для организации циклических алгоритмов используются конструкции

while <список команд 1> do <список команд 2> done

или

until <список команд 1> do <список команд 2> done

или

for <переменная> = <строка> do <список команд 2> done

или

for ((expr1; expr2; exprN)) do <список команд 2> done

В первой конструкции циклически выполняется список команд 2 до тех пор, пока код возврата последней команды из списка 1 остается равным 0.

Во второй конструкции выполняется список команд 2 до тех пор, пока код возврата последней команды из списка 1 остается не равным 0.

В третьей конструкции список команд 2 выполняется 1 раз для каждого значения переменной, которая последовательно приравнивается к очередному слову из *строки*.

В четвертой конструкции список команд выполняется по одному разу для каждого из указанных выражений.

Наиболее часто используемые команды в shell-процедурах

echo — вывод в файл stdout строки – параметра команды echo

shift — сдвиг влево позиционных параметров в теле процедуры

test или *[...]* — сравнение целых чисел, символьных строк, определение существования файлов

read — считывание из стандартного ввода символьной строки

break — прекращение выполнения цикла, переход к следующей за *done* команде

continue – переход к выполнению очередной итерации цикла

exit – прекращение выполнения процедуры и формирование кода возврата

expr – вычисление арифметических выражений, обработка символьных строк

Для повышения гибкости в shell-процедурах используются подстановки 3-х типов:

Подстановки команд, подстановки переменных, тильда-подстановки.

При подстановке команды в конструкции `<*команда*>` или $(<*команда*>) вместо

указанной команды подставляется результат ее выполнения.

При подстановке переменной в конструкции $<*переменная*> подставляется значение *переменной*.

При тильда-подстановке вместо символа ~ в символьную строку подставляется полное маршрутное имя домашнего каталога.

В shell-процедурах можно управлять реакцией (прерыванием) на сигналы, поступающие от различных процессов в системе, в том числе инициированные клавиатурой терминала или от других выполняющихся процедур.

Имеются 3 возможные реакции на сигналы:

1. Сигналы игнорируются (в этом случае сигнал вообще не посылается процессу).
2. Сигналы перехватываются (реакция на сигнал обеспечивается самим процессом).
3. Сигналы пропускаются без обработки. Работа процесса завершается.

Для управления прерываниями используется команда trap и команда kill.

trap – перехват сигнала, блокирование прерывания, восстановление стандартной реакции на сигнал

kill – передача сигнала процессу

Примеры использования команд:

trap 'rm /tmp/ps$$; exit' 2 — перехватывает сигнал 2 (прерывание) и выполняет команду *rm /tmp/ps$$; exit* — встроенная команда exit прерывает выполнение командного файла

Если при входе в командный файл сигнал игнорируется, то игнорируется и команда trap.

Если аргументом команды trap является пустая строка, то процедура игнорирует сигналы. В качестве примера можно привести фрагмент команды nohup:

```
trap '' 1 2 3 15
```

После выполнения этой команды как процедура, так и вызывающие ее команды будут игнорировать перечисленные сигналы.

Стандартная реакция на сигналы может быть восстановлена, например, следующей командой:

```
trap 2 3
```

Список текущих значений сигналов системы может быть получен с помощью команды

```
trap
```

Примеры shell-процедур

Пример 1

Написать процедуру, выводящую на экран текущую дату и время, полное маршрутное имя текущего каталога, полное маршрутное имя домашнего каталога пользователя, регистрационное имя пользователя, собственное имя процедуры.

Решение:

```
#proc1.1
#Вывод на экран текущей даты и времени
     date
#Вывод на экран полного маршрутного имени  текущего каталога
     pwd
#Вывод на экран полного маршрутного имени  домашнего каталога
пользователя
     echo $HOME
#Вывод на экран регистрационного имени  пользователя
     who am i
#Вывод на экран собственного имени процедуры
     echo $0
#Конец процедуры proc1.1
```

Пример выполнения процедуры:

```
$proc1.1
     /home/user1/posobie
     /home/user1
     user1    pts/1        2009-02-05 10:19
     proc1.1
```

Пример 2

Написать процедуру, выводящую на экран все переданные процедуре параметры: текущую дату и время, полное маршрутное имя текущего каталога, полное маршрутное имя домашнего каталога пользователя, регистрационное имя пользователя, собственное имя процедуры, количество переданных в процедуру параметров.

Решение:

```
#proc1.2
```

```
#Вывод на экран значений первого, второго, третьего и четвертого
позиционных
# параметров, переданных процедуре при обращении, а также
#нулевого параметра,
# содержащего имя выполняемой процедуры
     echo $1 $2 $3 $4 $0
#Конец процедуры proc1.2
```

Пример выполнения процедуры:

```
$proc1.2 `date` ` pwd` $HOME ` who am i`
     /home/user1/posobie user1 pts/1 2007-02-05 proc1.2
```

Пример 3

Написать процедуру, записывающую в некоторый файл текст, вводимый с клавиатуры при работе процедуры. Имя файла передается в процедуру в качестве параметра. В конце текста должна быть текущая дата.

Решение:

```
#proc2.3
     echo Введите текст
     read $TEMP1
     echo $TEMP1 >$1
     date >TEMP2
     cat $1 TEMP2 >TEMP1
     mv TEMP1 $1
#Конец процедуры proc2.3
```

Пример выполнения процедуры:

```
$proc2.3 fff
     Введите текст
     this text is test <ENTER>
$cat fff
     this text is test
     Сбт Фев 24 17:10:36 MSK 2009
```

Пример 4

Написать процедуру, вводящую в начало файла_1 некоторый текст с клавиатуры, затем присоединяющего содержимое файла_2, затем снова вводящего текст с клавиатуры в конец

файла_1. Имена файлов передаются в процедуру в качестве параметров.

Решение:

```
#proc2.4
     cat - $1 >TEMP1
     cat TEMP1 $2 >$1
#Конец процедуры proc2.4
```

Пример выполнения процедуры:

```
$cat fff
$proc2.4 fff ffff
$cat ffff
     +++++++++
     111111111111
     222222222222
     ----------------------
$
```

Пример 5

Написать процедуру, находящую в файле, имя которого передается в качестве параметра, все строки, содержащие вводимую с клавиатуры во время работы процедуры последовательность символов.

Решение:

```
#proc3.2
#Ввод с клавиатуры заданной последовательности символов и #запись
ее в переменную A
     read A
#Поиск в заданном файле всех строк, содержащих заданную
#последовательность
     grep $A $1
#Конец процедуры Proc3.2
```

Пример выполнения процедуры:

```
$ cat f1
     1234567890
     0987654321
     qwe456rtyu
```

```
     asd34567fg
     456456456
$ proc3.2 f1
     1234567890
     qwe456rtyu
     asd34567fg
     456456456
```

Пример 6

Написать процедуру, вводящую с клавиатуры два целых числа и выводящую на экран их сумму, произведение и частное от деления первого числа на второе.

Решение:

```
#proc4.1
#Ввод с клавиатуры двух целых чисел и присваивание их переменным A и B.
     read A B
#Вычисление суммы введенных чисел и присваивание ее переменной SUM
     SUM=`expr $A + $B`
#Вычисление произведения введенных чисел и присваивание результата переменной PROD
     PROD=`expr $A \* $B`
#Вычисление частного от деления первого введенного числа на
#второе и присваивание
# результата переменной QUOT
     QUOT =`expr $A \/ $B`
#Вывод на экран полученных результатов
     echo Сумма введенных чисел равна
     echo $SUM
     echo Произведение введенных чисел равно
     echo $PROD
     echo Частное от деления первого числа на второе равно
     echo $QUOT
#Конец процедуры Proc4.1
```

Пример выполнения процедуры:

```
$proc4.1
     4 5 <enter>
```

```
     Сумма введенных чисел равна
     9
     Произведение введенных чисел равно
     45
     Частное от деления первого числа на второе равно
     0
$
```

Пример 7

Написать процедуру, которая всем пользователям, работающим в данный момент в системе, посылает сообщение из заданного первым параметром файла.

Решение:

```
#proc5.2
     mail `who |cut -f1 -d" "` <$1
#Конец процедуры Proc5.2
```

Пример выполнения процедуры proc5.2:

```
$proc5.2 testing
$
```

Пример 8

Написать процедуру, которая выводит на экран постранично содержимое одного из двух символьных файлов, имена которых передаются процедуре в качестве параметров. Выбрать файл с меньшим числом строк (код завершения 000). Если заданные файлы имеют одинаковое число строк, то вывести на экран содержимое обоих файлов (код завершения 111).

Решение:

```
#proc6.1
#Запись в переменную TEMP1 числа строк в файле $1
     TEMP1=`wc -l $1 | cut -f1 -d " "`
#Запись в переменную TEMP2 числа строк в файле $2
     TEMP2=`wc -l $2 | cut -f1 -d " "`
#Выбор файла с меньшим числом строк
     if test $TEMP1 -lt $TEMP2
          then
```

```
#Вывод на экран содержимого файла $1
          more $1
#Выход из процедуры
          exit 000
     fi
     if test $TEMP2 -lt $TEMP1
          test
#Вывод на экран содержимого файла $2
          more $2
#Выход из процедуры
          exit 000
     fi
#Проверка равенства числа строк в файлах $1 и $2
     if test $TEMP1 –eq $TEMP2
          test
#Вывод на экран содержимого файлов $1 и $2
          more $1 $2
#Выход из процедуры
          exit 111
     fi
#Конец процедуры Proc6.1
```

Пример выполнения процедуры proc6.1:

```
$cat f1
     11111111111111111111
     22222222222222222222
     33333333333333333333
     44444444444444444444
$cat f2
     11111111111111111111
     22222222222222222222
$proc6.1 f1 f2
     1111111111111111111111111111
     2222222222222222222222222222
$
```

Пример 9

Написать процедуру, которая:

- среди пользователей, работающих в данный момент времени в системе, находит всех пользователей, имена которых содержатся во вводимой в процедуру символьной строке;
- выводит на экран найденные имена пользователей.

Решение:

```
#proc7.1
#формирование в файле file списка работающих пользователей
     who >file
#Ввод заданных имен пользователей
     echo "input string:"
     read temp
# Присваивание позиционным параметрам заданных в строке #имен пользователей
     set  $temp
# Поиск в файле file заданных имен и вывод их на экран
     while [ "$1" ]
     do
# Поиск в файле file очередного имени
          grep $1 file
          shift
     done
# Конец процедуры proc7.1
```

Пример выполнения процедуры:

```
$who
     user2    tty2         2007-03-12 15:52
     user3    tty3         2007-03-12 16:18
     user1    :0           2007-03-12 10:43 (console)
     user1    pts/0        2007-03-12 10:43
     user1    pts/1        2007-03-12 10:50
$proc7.1
     "input string:"
     user2 user3
     user2    tty2         2007-03-12 15:52
     user3    tty3         2007-03-12 16:18
```

```
$
```

Пример 10

Написать бесконечную процедуру, которая периодически, каждые 15 с, опрашивает систему и выводит на экран сведения о каждом вновь вошедшем в систему пользователе.

Решение:

```
#proc8.2
# Сохранение сведений о пользователях, работающих в системе #в момент запуска процедуры на выполнение
     who >file1
# Сохранение списка имен работающих пользователей
     cut –f1 -d=" "  file1 >file3
# Бесконечный цикл
     while true
     do
# Пауза 15 с
          sleep 15
# Сохранение сведений о работающих пользователях через 15 с
          who >file2
# Сохранение списка имен работающих пользователей
          cut –f1 -d=" "  file2 >file4
#Поиск каждого нового пользователя в списке старых пользователей и вывод на экран
# сведений о каждом вновь вошедшем в систему пользователе
          for newuser in file4
          do
               if grep $newuser file3 then
                    continue
               else
                    grep $newuser file2
               fi
          done
# Обновление сведений о  работающих  пользователях
          cp file4 file3
     done
# Конец процедуры Proc8.2
```

Пример выполнения процедуры:

```
$ proc8.2
     user3    tty2          2009-09-24-03-17 23:05
$
```

Пример 11. Пример использования команды trap

Написать shell- процедуру, которая:

- осуществляет просмотр подкаталогов в текущем каталоге и выполнение команд, вводимых с терминала, до получения сигнала **конец_файла** или **прерывание**. (Сигнал прерывание действует только по окончании выполнения команды).

Решение:

```
d=`pwd`
for i in *
do
     if test -d $d/$i
          then cd $d/$i
          while echo $i:
               trap exit 2
                    read x
               do trap : 2; eval $x; done
          fi
     done
# Встроенная команда read x считывает строки из файла стандартного ввода и #присваивает результат их выполнения переменной x.
#Сигналы конец_файла или прерывание прекращают выполнение этой команды с кодом #завершения, не равным нулю.
```

Индивидуальные задания к работе 5

Вариант 1

Написать shell-процедуру, которая:

- вводит передаваемое в качестве 1-го параметра количество символьных строк;
- в каждой введенной строке ищет подстроку, передаваемую в качестве второго параметра;
- заменяет каждую найденную подстроку на строку, передаваемую в качестве третьего параметра;
- выводит на экран каждую введенную строку и соответствующую ей новую строку.

Вариант 2

Написать shell-процедуру, которая:

- вводит 2 символьные строки;
- в каждой введенной строке ищет подстроку, передаваемую в качестве параметра;
- заменяет каждую найденную подстроку на пробел;
- образует из полученных строк третью строку так, чтобы в ней чередовались слова из первой и второй строк;
- выводит на экран введенные строки и новую строку.

Вариант 3

Написать shell-процедуру, которая:

- вводит символьную строку;
- во введенной строке ищет подстроку, передаваемую в качестве первого параметра;
- вставляет после каждой найденной подстроки символ, передаваемый в качестве второго параметра;
- удаляет из полученной строки символ, передаваемый в качестве третьего параметра;
- выводит на экран введенную и новую строку.

Вариант 4

Написать shell-процедуру, которая:

- вводит символьную строку;
- проверяет введенную строку на совпадение со строкой, переданной в качестве 1-го параметра;
- если строки совпадают, то выдает на экран приглашение повторить ввод;
- если не совпадают, то сравнивает длину введенной строки с длиной 2-го параметра, и, в случае их равенства, выводит на экран введенную строку в обратном порядке составляющих ее символов.

Вариант 5

Написать shell-процедуру, которая:

- вводит символьную строку;
- проверяет введенную строку на совпадение со строками, содержащимися в файле, имя которого передается в качестве 1-го параметра;
- для всех найденных совпадений заменяет соответствующие строки в файле на строку, переданную в качестве 2-го параметра;
- выводит на экран старое и новое содержимое файла, а также число найденных совпадений.

Вариант 6

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую маршрутное имя некоторого файла; проверяет введенное маршрутное имя, если оно начинается с символа /, на совпадение его первой части с маршрутным именем домашнего каталога пользователя;
- если введенное маршрутное имя содержит маршрутное имя домашнего каталога или является относительным, то проверяет существование указанного файла, в противном случае выводит на экран сообщение об ошибке;
- если файл существует, то выводит на экран его содержимое;
- если файл не существует, то создает его и записывает в него строку, передаваемую в качестве параметра.

Вариант 7

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую имя некоторого файла;
- проверяет наличие файла в домашнем каталоге или в одном из подкаталогов;
- если файл существует, то выводит на экран его содержимое;
- если файл не существует, то создает его и записывает в него с консоли некоторый текст;
- устанавливает для файла права доступа, передаваемые в качестве параметра.

Вариант 8

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую текст некоторого сообщения;
- проверяет наличие в своем почтовом ящике такого же сообщения;
- если в почтовом ящике имеется введенное сообщение, то выводит его на экран и посылает на терминалы всем пользователям, в данный момент работающим в системе из числа тех, чьи имена передаются в качестве параметров;
- всем остальным пользователям, чьи имена передаются в качестве параметров, рассылает введенное сообщение по почте.

Вариант 9

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую текст некоторого сообщения;
- проверяет регистрацию в системе пользователей, чьи имена переданы вторым и последующими параметрами;
- всем пользователям, чьи имена передаются в качестве второго и следующих параметров и работающим в системе в течение заданного первым параметром времени, рассылает введенное сообщение по почте;
- всем остальным пользователям, работающим в данный момент в системе, рассылает прямые сообщения, содержащие введенную символьную строку.

Вариант 10

Написать shell-процедуру, которая:

- проверяет свой почтовый ящик на наличие в нем сообщений;
- находит в почтовом ящике одинаковые по тексту сообщения;
- всем пользователям, приславшим более одного сообщения с одинаковым текстом, рассылает по почте сообщения, текст которого содержится в файле, имя которого передается в качестве параметра.

Вариант 11

Написать shell-процедуру, которая:

- удаляет из заданного первым параметром каталога и всех подкаталогов файлы, дата последней модификации которых предшествует текущей дате минус число дней, переданное в качестве второго параметра;
- изменяет дату последней модификации всех остальных файлов указанного каталога на текущую без изменения содержимого файлов.

Вариант 12

Написать shell-процедуру, которая:

- выводит на экран список всех пользователей системы, включенных в заданную первым параметром группу пользователей;
- для каждого из заданных третьим и следующими параметрами имен пользователей выводит на экран права доступа к заданному вторым параметром файлу.

Вариант 13

Написать shell-процедуру, которая:

- всем пользователям, работающим в данный момент в системе, имена которых задаются третьим и последующими параметрами, рассылает сообщения из файла, имя которого передается вторым параметром;
- повторяет сообщения с периодичностью, задаваемой первым параметром;
- прекращает выдачу сообщений при вводе слова **quit.**

Вариант 14

Написать shell-процедуру, которая:

- всем пользователям, не включенным в заданную первым параметром группу, посылает по почте сообщения о текущем времени и дате;
- всем пользователям, включенным в заданную первым параметром группу, посылает по почте сообщения Happy New Year!.

Вариант 15

Написать shell-процедуру, которая:

- вводит с терминала некоторое целое число;
- всем пользователям, работающим в данный момент в системе, посылает сообщение о числе порожденных ими процессов;
- тем пользователям, у которых число процессов больше введенного числа, посылает второе сообщение с предупреждением.

Вариант 16

Написать shell-процедуру, которая:

- вводит с терминала некоторое целое число;
- всем пользователям, работающим в данный момент в системе, посылает сообщение о среднем числе процессов у каждого пользователя ;
- тем пользователям, у которых число процессов больше среднего, посылает второе сообщение с предупреждением, если число процессов у них отличается от среднего больше, чем введенное целое число.

Вариант 17

Написать shell-процедуру, которая:

- все почтовые сообщения, полученные от заданного первым параметром пользователя, посылает работающим в данный момент в системе пользователям;
- если в системе работает пользователь, приславший данное сообщение, то этому пользователю посылает сообщение, вводимое с терминала.

Вариант 18

Написать shell-процедуру, которая:

- всем пользователям, у которых есть больше 1 приостановленного процесса, посылает на терминал предупредительное сообщение;
- если пользователь, получивший предупредительное сообщение, в течение 2 минут не ответит и не уменьшит число приостановленных процессов, то его имя записывается в заданный файл.

Вариант 19

Написать shell-процедуру, которая:

- периодически проверяет содержимое некоторого файла, хранящего имена пользователей, работавших в системе в заданный интервал времени;
- если в файле имя пользователя встречается более 2-х раз, то такому пользователю выдается некоторое сообщение, текст которого передается в качестве второго параметра;
- если требуемый пользователь в данный момент не работает в системе, то ему передается сообщение по почте.

Вариант 20

Написать shell-процедуру, которая:

- в заданном первым параметром каталоге находит все простые файлы, число ссылок на которые максимально, и удаляет их;
- удаляет все пустые каталоги;
- выдает на экран сообщения о каждом удаленном файле и каталоге.

Вариант 21

Написать shell-процедуру, которая:

- в заданном первым параметром каталоге находит все подкаталоги, число простых файлов в которых больше заданного вторым параметром числа
- удаляет найденные подкаталоги;
- выдает на экран сообщения о каждом удаленном каталоге.

Вариант 22

Написать shell-процедуру, которая:

- в заданном первым параметром каталоге находит все простые файлы, в которых содержится заданная вторым параметром символьная строка;
- в найденных файлах удаляет все повторяющиеся строки;
- выводит на экран имена всех найденных файлов.

Вариант 23

Написать shell-процедуру, которая:

- в каталоге, имя которого передается первым параметром, находит все простые файлы размером более заданного вторым параметром;
- создает в указанном каталоге 3 новых каталога;
- помещает в созданные каталоги файлы из исходного каталога: в первый – файлы, содержащие одну строку с заданным словом, во второй – файлы с двумя такими строками, в третий – с тремя;
- имена всех файлов, не включенных в новые каталоги, выводит на экран.

Вариант 24

Написать shell-процедуру, которая:

- в заданном первым параметром каталоге находит все простые файлы, в которых содержатся заданные вторым и третьим параметрами символьные строки;
- в найденных файлах удаляет все повторяющиеся строки;
- выводит на экран имена всех полученных файлов.

Вариант 25

Написать shell-процедуру, которая:

- находит в текущем каталоге все файлы, имена которых вводятся при работе процедуры по запросу, выводимому на экран;
- в каждом найденном файле ищет строку, содержащую слово, переданное первым параметром;
- если такая строка имеется в файле, то на экран выводится ее номер (или номера, если таких строк несколько);
- при отсутствии в файле таких строк выводит соответствующее сообщение.

Вариант 26

Написать shell-процедуру, которая:

- вычисляет значение арифметического выражения, заданного первыми 7 параметрами;
- сравнивает полученное значение с числом, вводимым при исполнении процедуры;
- при совпадении результатов сравнения выводит на экран заданное выражение и его значение.

Вариант 27

Написать shell-процедуру, которая:

- в заданном первым параметром файле находит все строки-омонимы;
- выводит на экран найденные строки;
- подсчитывает в каждой строке число символов, совпадающих с заданным вторым параметром символом, и выводит его на экран.

Вариант 28

Написать shell-процедуру, которая:

- определяет высоту поддерева каталогов, начиная от каталога, передаваемого в качестве первого параметра;
- выводит на экран полное маршрутное имя каталога, последнего в ветви поддерева максимальной длины.

Вариант 29

Написать shell-процедуру, которая:

- среди пользователей, работающих в данный момент времени в системе, находит пользователей, имена которых содержатся в файле, передаваемом в качестве первого

параметра;

- выводит на экран найденные имена пользователей;
- тем пользователям, имена которых вводятся при исполнении процедуры, передает сообщение, текст которого содержится в файле (имя файла передается в качестве второго параметра).

Вариант 30

Написать shell-процедуру, которая:

- читает содержимое первого файла, передаваемого в качестве первого параметра;
- выводит на экран каждые n секунд banner, текст которого содержится во втором файле, имя которого задается вторым параметром;
- очередное значение n содержится в очередной строке первого файла.

Вариант 31

Написать shell-процедуру, которая:

- читает содержимое первого файла, передаваемого в качестве первого параметра;
- читает содержимое второго файла, передаваемого в качестве второго параметра;
- находит в первом файле строку, содержащую заданное третьим параметром слово;
- вставляет содержимое второго файла после найденной строки первого файла;
- выводит на экран содержимое полученного файла.

Вариант 32

Написать shell-процедуру, которая:

- читает содержимое первого файла, передаваемого в качестве первого параметра;
- читает содержимое второго файла, передаваемого в качестве второго параметра;
- если число строк в первом и втором файлах одинаковое, то выводит на экран каждые 5 секунд попеременно строки из первого и второго файлов.

Вариант 33

Написать shell-процедуру, которая:

- читает содержимое первого файла, передаваемого в качестве первого параметра;
- читает содержимое второго файла, передаваемого в качестве второго параметра;
- выводит на экран каждые 7 секунд попеременно 2 строки из первого и 1 строку из второго файла, перемещаясь по файлам циклически.

Вариант 34

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую некоторое целое число;
- читает содержимое файла, передаваемого в качестве первого параметра;

- выводит на экран каждые 6 секунд попеременно 2 строки из файла и 1 введенную строку, перемещаясь по файлу циклически.

Вариант 35

Написать shell-процедуру, которая:

- вводит символьную строку;
- читает содержимое файла, передаваемого в качестве первого параметра;
- выводит на экран каждые 6 секунд попеременно 2 строки из файла и 1 введенную строку, перемещаясь по файлу циклически.

Вариант 36

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую два целых числа m и n, разделенных пробелами;
- читает содержимое файла, передаваемого в качестве первого параметра;
- выводит на экран каждые 5 секунд попеременно m строк из файла и n строк "Будь здоров".

Вариант 37

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую два целых числа m и n, разделенных пробелами;
- читает содержимое файла, передаваемого в качестве первого параметра;
- выводит на экран каждые m секунд попеременно n строк из файла и пустую строку.

Вариант 38

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую три целых числа k,m и n, разделенных пробелами;
- читает содержимое файла, передаваемого в качестве первого параметра;
- выводит на экран каждые m секунд попеременно m строк из файла и n пустых строк.

Вариант 39

Написать shell-процедуру, которая:

- читает содержимое файла, передаваемого в качестве первого параметра;
- записывает через каждые 5 секунд в файл, имя которого передается в качестве второго параметра, попеременно строки из первого файла и текущее время и дату;
- выводит на экран каждые 7 секунд текущее содержимое второго файла;
- при вводе с клавиатуры слова quit удаляет второй файл и завершает работу.

Вариант 40

Написать shell-процедуру, которая:

- читает содержимое файла, передаваемого в качестве первого параметра;
- создает новый файл, имя которого передается в качестве второго параметра;
- выводит на экран каждые 7 секунд очередную строку первого файла;
- сортирует все выведенные на экран строки первого файла по длине и записывает их в новый файл;
- при вводе с клавиатуры слова quit удаляет новый файл и завершает работу.

Вариант 41

Написать shell-процедуру, которая:

- читает содержимое файла, имя которого вводится при исполнении процедуры;
- создает новый файл, имя которого передается в качестве параметра;
- выводит на экран каждые 6 секунд очередные 2 строки файла;
- сортирует выведенные на экран строки по длине и записывает их в новый файл;
- при вводе с клавиатуры слова end удаляет второй файл и завершает работу.

Вариант 42

Написать shell-процедуру, которая:

- читает содержимое файла, передаваемого в качестве первого параметра;
- создает новый файл, имя которого передается в качестве второго параметра;
- записывает в новый файл строки первого файла в обратном порядке, вставляя после каждого слова фразу "THAT IS ALL" столько раз , сколько задано третьим параметром.

Вариант 43

Написать shell-процедуру, которая:

- читает содержимое файла, передаваемого в качестве первого параметра;
- создает в текущем каталоге цепочку подкаталогов с относительным маршрутным именем, повторяющим полное маршрутное имя текущего каталога;
- создает в последнем подкаталоге новый файл, имя которого передается в качестве второго параметра;
- записывает в новый файл строки первого файла в обратном порядке, вставляя после каждого слова фразу "THAT IS ALL" столько раз, сколько задано третьим параметром.

Вариант 44

Написать shell-процедуру, которая:

- читает содержимое файла, передаваемого в качестве первого параметра;
- создает в текущем каталоге цепочку подкаталогов с относительным маршрутным именем, повторяющим полное маршрутное имя текущего каталога;

- создает в последнем подкаталоге новый файл, имя которого передается в качестве второго параметра;
- записывает в новый файл строки первого файла в обратном порядке, вставляя после каждого слова фразу "THAT IS ALL" столько раз, сколько задано третьим параметром.

Вариант 45

Написать shell-процедуру, которая:

- читает содержимое файла, передаваемого в качестве первого параметра (в файле в каждой строке представлено одно целое число);
- проверяет правильность формата содержимого файла, при ошибках выводит соответствующие сообщения и завершает работу;
- подсчитывает сумму всех содержащихся в файле чисел;
- выводит на экран полученную сумму.

Вариант 46

Написать shell-процедуру, которая:

- читает содержимое файла, передаваемого в качестве первого параметра (в файле в каждой строке представлено по два целых числа, разделенных пробелами);
- проверяет правильность формата содержимого файла, при ошибках выводит соответствующие сообщения и завершает работу;
- подсчитывает сумму чисел в каждой строке файла;
- сортирует полученные суммы по убыванию и выводит их на экран.

Вариант 47

Написать shell-процедуру, которая:

- читает содержимое двух файлов, передаваемых в качестве первых параметров (в файлах в каждой строке представлено по одному целому числу);
- проверяет правильность формата содержимого файлов, при ошибках выводит соответствующие сообщения и завершает работу;
- выбирает одинаковые числа в первом и втором файлах;
- сортирует полученные числа по возрастанию и выводит их на экран.

Вариант 48

Написать shell-процедуру, которая:

- читает содержимое двух файлов, передаваемых в качестве первых параметров (в файлах в каждой строке представлено по одному целому числу);
- проверяет правильность формата содержимого файлов, при ошибках выводит соответствующие сообщения и завершает работу;
- суммирует числа из первого и второго файлов, расположенные в строках с одинаковым номером;

- сортирует полученные суммы по возрастанию и выводит их на экран

Вариант 49

Написать shell-процедуру, которая:

- читает содержимое двух файлов, передаваемых в качестве первых параметров (в файлах в каждой строке представлено не более, чем по три целых числа, разделенных пробелами);
- проверяет правильность формата содержимого файлов, при ошибках выводит соответствующие сообщения и завершает работу;
- вычисляет суммы чисел в каждой строке;
- сортирует полученные положительные суммы по возрастанию и выводит их на экран.

Вариант 50

Написать shell-процедуру, которая:

- читает содержимое трех файлов, передаваемых в качестве первых параметров (в файлах в каждой строке представлено не более, чем по четыре целых числа, разделенных пробелами);
- проверяет правильность формата содержимого файлов, при ошибках выводит соответствующие сообщения и завершает работу;
- вычисляет произведения чисел в каждой строке;
- сортирует полученные произведения по возрастанию и выводит их на экран.

Вариант 51

Написать shell-процедуру, которая:

- вводит передаваемое в качестве 1-го параметра количество символьных строк;
- в каждой введенной строке ищет слово, передаваемую в качестве второго параметра;
- заменяет каждую найденную подстроку на пустую строку;
- выводит на экран каждую введенную строку и ее номер.

Вариант 52

Написать shell-процедуру, которая:

- вводит 3 символьные строки;
- в каждой введенной строке ищет подстроку, передаваемую в качестве параметра;
- заменяет каждую найденную подстроку на пробел;
- образует из полученных строк четвертую строку так, чтобы в ней чередовались слова из первой, второй и третьей строк;
- выводит на экран введенные строки и новую строку.

Вариант 53

Написать shell-процедуру, которая:

- вводит символьную строку;
- во введенной строке ищет подстроку, передаваемую в качестве первого параметра;
- вставляет перед каждой найденной подстрокой группу символов, передаваемых в качестве второго параметра;
- удаляет из полученной строки группу символов, передаваемых в качестве третьего параметра;
- выводит на экран введенную и новую строку.

Вариант 54

Написать shell-процедуру, которая:

- вводит символьную строку;
- проверяет введенную строку на несовпадение со строкой, переданной в качестве 1-го параметра;
- если строки не совпадают, то выдает на экран приглашение повторить ввод;
- если совпадают, то сравнивает длину введенной строки с длиной 2-го параметра, и, в случае их равенства, выводит на экран введенную строку.

Вариант 55

Написать shell-процедуру, которая:

- вводит символьную строку;
- проверяет введенную строку на несовпадение со строками, содержащимися в файле, имя которого передается в качестве 1-го параметра;
- для всех найденных несовпадений заменяет соответствующие строки в файле на пустые строки;
- выводит на экран старое и новое содержимое файла, а также число найденных несовпадений.

Вариант 56

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую маршрутное имя некоторого файла;
- проверяет введенное маршрутное имя, если оно начинается с символа /, на совпадение его первой части с маршрутным именем домашнего каталога пользователя;
- если введенное маршрутное имя содержит маршрутное имя домашнего каталога или является относительным, то проверяет существование указанного первым параметром файла, в противном случае выводит на экран сообщение об ошибке;
- если файл существует, то выводит на экран его содержимое в обратном порядке;
- если файл не существует, то создает его и записывает в него строку, передаваемую в качестве параметра в обратном порядке.

Вариант 57

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую имя некоторого каталога;
- проверяет наличие каталога в домашнем каталоге или в одном из подкаталогов;
- если каталог существует, то выводит на экран его содержимое;
- если каталог не существует, то создает его;
- устанавливает для каталога права доступа, соответствующие правам доступа к указанному каталогу.

Вариант 58

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую текст некоторого сообщения;
- проверяет наличие в своем почтовом ящике хотя бы одного сообщения того же автора;
- если в почтовом ящике имеются такие сообщения, то выводит их на экран и посылает эти сообщения на терминалы всем пользователям, в данный момент работающим в системе из числа тех, чьи имена передаются в качестве параметров.

Вариант 59

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую имя файла с текстом некоторого сообщения;
- всем пользователям, чьи имена передаются в качестве второго и следующих параметров и работающим в системе в течение заданного первым параметром времени, рассылает введенное сообщение по почте;
- всем остальным пользователям, работающим в данный момент в системе, рассылает прямые сообщения, содержащие введенную символьную строку.

Вариант 60

Написать shell-процедуру, которая:

- проверяет свой почтовый ящик на отсутствие в нем сообщений;
- создает некоторое сообщение, текст которого помещает в файл, имя которого передается в качестве первого параметра;
- всем пользователям, работающим в системе, рассылает по почте сообщения, текст которого содержится в файле, имя которого передается в качестве второго параметра.

Вариант 61

Написать shell-процедуру, которая:

- удаляет из заданного первым параметром каталога, дата последней модификации которых предшествует текущей дате, переданной в качестве второго параметра;
- изменяет дату последней модификации всех остальных файлов указанного каталога на текущую без изменения содержимого файлов;
- выводит на экран имена всех модифицированных файлов.

Вариант 62

Написать shell-процедуру, которая:

- выводит на экран список всех пользователей системы, включенных в заданные первым и вторым параметрами группы пользователей;
- для заданного третьим параметром имени пользователя выводит на экран права доступа к заданному четвертым параметром файлу.

Вариант 63

Написать shell-процедуру, которая:

- всем пользователям, работающим в данный момент в системе, имена которых задаются третьим и последующими параметрами, рассылает сообщения, текст которого вводится с клавиатуры при исполнении процедуры;
- повторяет сообщения с периодичностью, задаваемой первым параметром;
- прекращает выдачу сообщений при вводе слова quit и прекращает свою работу.

Вариант 64

Написать shell-процедуру, которая:

- всем пользователям, не включенным во вводимую с клавиатуры при исполнении прцедуры группу, посылает по почте сообщения о текущем времени и дате;
- всем пользователям, включенным в заданную первым параметром группу, посылает по почте сообщения Happy New Year!;
- выводит на экран имена пользователей, которым послано сообщение.

Вариант 65

Написать shell-процедуру, которая:

- вводит с терминала некоторое целое положительное число;
- всем пользователям, работающим в данный момент в системе, посылает сообщение о числе порожденных ими процессов и допустимом числе, введенном с терминала;
- тем пользователям, у которых число процессов больше введенного числа, посылает второе сообщение с предупреждением и выводит их имена на экран.

Вариант 66

Написать shell-процедуру, которая:

- вводит с терминала некоторое целое число;
- всем пользователям, работающим в данный момент в системе, посылает сообщение о среднем числе процессов у каждого пользователя;
- тем пользователям, у которых число процессов больше среднего, посылает второе сообщение с предупреждением, если число процессов у них отличается от среднего больше, чем заданное первым параметром число.

Вариант 67

Написать shell-процедуру, которая:

- все почтовые сообщения, полученные от заданного первым параметром пользователя, посылает работающим в данный момент в системе пользователям;
- если в системе работает пользователь, приславший данное сообщение, то этому пользователю посылает сообщение, содержащееся в файле, имя которого передается в качестве параметра.

Вариант 68

Написать shell-процедуру, которая:

- всем пользователям, у которых есть больше n приостановленых[процессов, посылает на терминал предупредительное сообщение;
- если пользователь, получивший предупредительное сообщение, в течение m минут не ответит и не уменьшит число приостановленных процессов, то его имя записывается в заданный файл. Вличины n и m передаются в качестве параметров.

Вариант 69

Написать shell-процедуру, которая:

- периодически проверяет содержимое некоторого файла, хранящего имена пользователей, не работавших в системе в заданный интервал времени;
- если в файле имя пользователя встречается более m раз, то такому пользователю выдается некоторое сообщение, текст которого передается в качестве второго параметра (величина m передается в качестве первого параметра);
- если требуемый пользователь в данный момент не работает в системе, то ему передается сообщение по почте.

Вариант 70

Написать shell-процедуру, которая:

- в заданном первым параметром каталоге находит все простые файлы, число ссылок на которые минимально, и удаляет их;
- удаляет все пустые каталоги;
- выдает на экран сообщения о каждом удаленном файле и каталоге.

Вариант 71

Написать shell-процедуру, которая:

- в заданном первым параметром каталоге находит все подкаталоги, число простых файлов в которых меньше или равно заданному вторым параметром числу;
- удаляет найденные подкаталоги;
- выдает на экран сообщения о каждом удаленном каталоге.

Вариант 72

Написать shell-процедуру, которая:

- в заданном первым параметром каталоге находит все простые файлы, в которых не содержится заданная вторым параметром символьная строка;
- в найденных файлах удаляет все повторяющиеся строки;

- выводит на экран содержимое всех найденных файлов;

Вариант 73

Написать shell-процедуру, которая:

- в каталоге, имя которого вводится с клавиатуры по запросу , находит все простые файлы размером более заданного вторым параметром;
- создает в указанном каталоге 3 новых каталога;
- помещает в созданные каталоги файлы из исходного каталога: в первый – файлы, не содержащие строкк с заданным словом, во второй – файлы с одной такой строкой, в третий – с двумя;
- имена всех файлов, не включенных в новые каталоги, выводит на экран.

Вариант 74

Написать shell-процедуру, которая:

- в заданном первым параметром каталоге находит все простые файлы, в которых содержатся заданные вторым или третьим параметрами символьные строки;
- в найденных файлах удаляет все повторяющиеся более 2-х раз строки;
- выводит на экран имена всех полученных файлов.

Вариант 75

Написать shell-процедуру, которая:

- находит в заданном каталоге все файлы, имена которых вводятся при работе процедуры по запросу, выводимому на экран;
- в каждом найденном файле ищет строку, не содержащую слово, переданное первым параметром;
- если такая строка имеется в файле, то на экран выводится ее номер (или номера, если таких строк несколько);
- при отсутствии в файле таких строк выводит соответствующее сообщение.

Вариант 76

Написать shell-процедуру, которая:

- вычисляет значение арифметического выражения, заданного первыми 5 параметрами;
- сравнивает полученное значение с числом, заданным вторым параметром;
- при совпадении результатов сравнения выводит на экран заданное выражение и его значение.

Вариант 77

Написать shell-процедуру, которая:

- в заданном первым параметром файле находит все слова-омонимы;
- выводит на экран строки, содержащие омонимы;
- подсчитывает в каждой строке число символов, совпадающих с заданным вторым

параметром символом, и выводит его на экран

Вариант 78

Написать shell-процедуру, которая:

- определяет высоту поддерева каталогов, начиная от каталога, передаваемого в качестве первого параметра;
- выводит на экран полное маршрутное имя каталога, последнего в ветви поддерева максимальной длины.

Вариант 79

Написать shell-процедуру, которая:

- среди пользователей, работающих в данный момент времени в системе, находит пользователей, имена которых содержатся в файлеимя которого вводится с клавиатуры;
- выводит на экран найденные имена пользователей;
- тем пользователям, имена которых передаются в качестве параметров процедуры, передает сообщение, текст которого содержится в файле (имя файла вводится при работе процедуры).

Вариант 80

Написать shell-процедуру, которая:

- читает содержимое первого файла, передаваемого в качестве первого параметра;
- выводит на экран каждые n секунд banner, текст которого содержится во втором файле, имя которого задается вторым параметром;
- очередное значение n содержится в очередной строке первого файла.

Вариант 81

Написать shell-процедуру, которая:

- читает содержимое первого файла, передаваемого в качестве первого параметра;
- читает содержимое второго файла, передаваемого в качестве второго параметра;
- находит во втором файле строку, содержащую заданное третьим параметром слово;
- вставляет содержимое первого файла перед найденной строкой второго файла;
- выводит на экран содержимое полученного файла.

Вариант 82

Написать shell-процедуру, которая:

- читает содержимое первого файла, передаваемого в качестве первого параметра;
- читает содержимое второго файла, передаваемого в качестве второго параметра;
- если число строк в первом и втором файлах различное, то выводит на экран каждые 10 секунд попеременно строки из первого и второго файлов.

Вариант 83

Написать shell-процедуру, которая:

- читает содержимое первого файла, передаваемого в качестве первого параметра;
- читает содержимое второго файла, передаваемого в качестве второго параметра;
- выводит на экран каждые 7 секунд попеременно 2 строки из первого и 1 строку из второго файла, перемещаясь по файлам циклически в обратном порядке.

Вариант 84

Написать shell-процедуру, которая:

- вводит символьные строки, содержащие некоторые целые числа;
- читает содержимое файла, передаваемого в качестве первого параметра;
- выводит на экран каждые 6 секунд попеременно 1 строку из файла и 2 введенные строки, перемещаясь по файлу циклически

Вариант 85

Написать shell-процедуру, которая:

- вводит символьную строку;
- читает содержимое файла, передаваемого в качестве первого параметра;
- выводит на экран каждые 6 секунд попеременно 2 строки из файла и дважды повторенную введенную строку, перемещаясь по файлу циклически.

Вариант 86

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую два целых числа m и n, разделенных знаком @;
- читает содержимое файла, передаваемого в качестве первого параметра;
- выводит на экран каждые 5 секунд попеременно n строк из файла и m пустых строк .

Вариант 87

Написать shell-процедуру, которая:

- вводит 2 символьные строки, содержащие каждая два целых числа, разделенных пробелами;
- читает содержимое файла, передаваемого в качестве первого параметра;
- выводит на экран каждые m секунд попеременно n строк из файла и пустую строку (число m есть разность первого и второго чисел в первой введенной строке, а число n – во второй.

Вариант 88

Написать shell-процедуру, которая:

- вводит символьную строку, содержащую три целых числа k,m и n, разделенных знаками @;
- читает содержимое файла, передаваемого в качестве первого параметра;
- выводит на экран каждые k секунд попеременно m строк из файла и n пустых строк.

Вариант 89

Написать shell-процедуру, которая:

- читает содержимое файла, передаваемого в качестве первого параметра;
- записывает через каждые 7 секунд в файл, имя которого передается в качестве второго параметра, попеременно 2 строки из первого файла и текущее время и дату;
- выводит на экран каждые 5 секунд текущее содержимое второго файла;
- при вводе с клавиатуры слова quit удаляет второй файл и завершает работу

Вариант 90

Написать shell-процедуру, которая:

- читает содержимое файла, передаваемого в качестве первого параметра;
- создает новый файл, имя которого передается в качестве второго параметра;
- выводит на экран каждые 7 секунд очередную строку первого файла;
- сортирует выведенные на экран строки и записывает их в новый файл;
- при вводе с клавиатуры дважды слова quit удаляет второй файл и завершает работу, одновременно выдавая на экран соответствующее сообщение.

Вариант 91

Написать shell-процедуру, которая:

- читает содержимое файла, имя которого вводится при исполнении процедуры;
- создает новый файл, имя которого передается в качестве параметра;
- выводит на экран каждые 6 секунд очередные 2 строки файла в обратном порядке;
- сортирует выведенные на экран строки по длине и записывает их в новый файл;
- при вводе с клавиатуры трижды слова end удаляет второй файл и завершает работу.

Вариант 92

Написать shell-процедуру, которая:

- читает содержимое файла, передаваемого в качестве первого параметра;
- создает новый файл, имя которого передается в качестве второго параметра;
- записывает в новый файл строки первого файла в обратном порядке, вставляя после каждой пары строк фразу "THAT IS ALL".

Вариант 93

Написать shell-процедуру, которая:

- читает содержимое файла, передаваемого в качестве первого параметра;
- создает в текущем каталоге цепочку подкаталогов с относительным маршрутным именем, повторяющим полное маршрутное имя текущего каталога;
- создает в последнем подкаталоге новый файл, имя которого передается в качестве второго параметра;
- записывает в новый файл строки первого файла в обратном порядке, вставляя после каждого слова фразу "THAT IS ALL" столько раз, сколько задано третьим параметром.

Вариант 94

Написать shell-процедуру, которая:

- читает содержимое файла, передаваемого в качестве первого параметра;
- создает в текущем каталоге цепочку подкаталогов с относительным маршрутным именем, повторяющим полное маршрутное имя текущего каталога;
- создает в последнем подкаталоге новый файл, имя которого передается в качестве второго параметра;
- записывает в новый файл строки первого файла попеременно в прямом и в обратном порядке, вставляя после каждого слова фразу "THAT IS ALL" столько раз, сколько задано третьим параметром.

Вариант 95

Написать shell-процедуру, которая:

- читает содержимое файла, передаваемого в качестве первого параметра (в файле в каждой строке представлено одно целое число);
- проверяет правильность формата содержимого файла, при ошибках выводит соответствующие сообщения и завершает работу;
- подсчитывает сумму всех содержащихся в файле чисел;
- выводит на экран исходные числа и полученную сумму.

Вариант 96

Написать shell-процедуру, которая:

- читает содержимое файла, передаваемого в качестве первого параметра (в файле в каждой строке представлено по три целых числа, разделенных пробелами);
- проверяет правильность формата содержимого файла, при ошибках выводит соответствующие сообщения и завершает работу;
- подсчитывает сумму чисел в каждой строке файла;
- сортирует полученные суммы по возрастанию и выводит их на экран;
- подсчитывает и выводит на экран общую сумму.

Вариант 97

Написать shell-процедуру, которая:

- читает содержимое двух файлов, передаваемых в качестве первых параметров (в файлах в каждой строке представлено по одному целому числу);
- проверяет правильность формата содержимого файлов, при ошибках выводит соответствующие сообщения и завершает работу;
- выбирает не совпадающие в одноименных строках числа в первом и втором файлах;
- сортирует полученные числа по возрастанию и выводит их на экран.

Вариант 98

Написать shell-процедуру, которая:

- читает содержимое двух файлов, передаваемых в качестве первых параметров (в файлах в каждой строке представлено по одному целому числу);
- проверяет правильность формата содержимого файлов, при ошибках выводит соответствующие сообщения и завершает работу;
- суммирует числа из первого и второго файлов, расположенные в строках с одинаковым номером;
- сортирует полученные суммы по возрастанию и выводит их на экран;
- подсчитывает общую сумму и выводит ее на экран.

Вариант 99

Написать shell-процедуру, которая:

- читает содержимое трех файлов, передаваемых в качестве первых параметров (в файлах в каждой строке представлено не более, чем по четыре целых числа, разделенных пробелами);
- проверяет правильность формата содержимого файлов, при ошибках выводит соответствующие сообщения и завершает работу;
- вычисляет суммы и произведения чисел в каждой строке;
- сортирует полученные суммы и произведения по возрастанию и выводит их на экран.

Вариант 100

Написать shell-процедуру, которая:

- читает содержимое трех файлов, передаваемых в качестве первых параметров (в файлах в каждой строке представлено не более, чем по два целых числа, разделенных знаком @);
- проверяет правильность формата содержимого файлов, при ошибках выводит соответствующие сообщения и завершает работу;
- вычисляет остаток от деления первого на второе число в каждой строке;
- сортирует полученные остатки по убыванию и выводит их на экран.

Вариант 101

Написать shell-процедуру, которая:

- ожидает ввода с клавиатуры сигнала < ctrl+\>;

- удаляет по этому сигналу из файла, имя которого передается в качестве 1-го параметра, заданное вторым параметром количество строк;
- ожидает ввода с клавиатуры сигнала <ctrl+c>;
- удаляет по этому сигналу первую и последнюю строки из файла, имя которого вводится с клавиатуры;
- по сигналу <ctrl+y> завершает работу.

Вариант 102

Написать shell-процедуру, которая:

- ожидает ввода с клавиатуры сигнала, имя которого передается в качестве 1-го параметра;
- читает по принятому сигналу из файла, имя которого передается в качестве второго параметра, последнюю строку;
- удаляет по этому же сигналу первую и последнюю строки из файла, имя которого вводится с клавиатуры, и заканчивает на этом свою работу.

Вариант 103

Написать shell-процедуру, которая:

- читает из стандартного ввода имя сигнала;
- читает по принятому сигналу из файла, имя которого передается в качестве первого параметра, последнюю строку;
- находит в прочитанной строке имена нескольких сигналов, по каждому из которых завершает свою работу, формируя соответствующий код завершения (для каждого сигнала свой код завершения).

Вариант 104

Написать 2 shell-процедуры.

Первая процедура:

- по заданному первым параметром сигналу запускает в фоновом режиме вторую процедуру;
- по заданному вторым параметром сигналу посылает во вторую процедуру сигнал INT и прекращает свою работу.

Вторая процедура:

- по сигналу INT, посланному из первой процедуры, удваивает содержимое файла, имя которого передается в первую процедуру в качестве третьего параметра, и на этом прекращает свою работу.

Вариант 105

Написать 2 shell-процедуры.

Первая процедура:

- вводит с клавиатуры имя сигнала;
- ожидает ввода с клавиатуры заданного сигнала;
- каждый раз при получении заданного сигнала с клавиатуры запускает в фоновом режиме вторую процедуру;
- по истечении заданного интервала завершает свою работу.

Вторая процедура:

- создает в домашнем каталоге новый подкаталог, имя которого передается в качестве очередного параметра первой процедуры, и на этом заканчивает свою работу.

Вариант 106

Написать 3 shell-процедуры.

Первая процедура:

- вводит целое число n;
- ожидает сигналов <ctrl+c>, <ctrl+y> или <ctrl+\>:
  - при получении сигнала <ctrl+c> запускает в фоновом режиме процедуру 2;
  - при получении сигнала <ctrl+y> запускает в фоновом режиме процедуру 3;
  - при получении сигнала <ctrl+\> посылает сигнал INT процедурам 2 и 3.

Вторая процедура:

- через каждые 5 сек. читает n раз содержимое файла, имя которого передается в качестве первого параметра процедуре 1, и добавляет его к содержимому файла, имя которого передается в качестве второго параметра процедуре 1, после чего заканчивает свою работу.

Третья процедура:

- удаляет файлы, имена которых переданы в качестве параметров первой процедуре, после чего заканчивает свою работу.

Вариант 107

Написать 3 shell-процедуры.

Первая процедура:

- ожидает сигнала <ctrl+\> или <ctrl+c>;
- при получении сигнала <ctrl+\> запускает в фоновом режиме процедуру 2;
- при получении сигнала <ctrl+c> по истечении 20 сек. посылает сигнал INT процедуре 2.

Вторая процедура:

- читает содержимое файла, имя которого передается в качестве первого параметра процедуре 1, и добавляет его к содержимому файла, имя которого передается в качестве второго параметра процедуре 1;

- запускает в фоновом режиме процедуру 3, после чего завершает свою работу

Третья процедура:

- по истечении 30 сек. посылает процедуре 1 сигнал <ctrl+c> и завершает работу.

Вариант 108

Написать 3 shell-процедуры.

Первая процедура:

- запускает в фоновом режиме процедуру 2;
- передает процедуре 2 имя сигнала 1;
- ожидает от процедуры 2 сигнала 2;
- при получении сигнала 2 запускает с задержкой 30 сек в фоновом режиме процедуру 3, после чего завершает работу.

Вторая процедура:

- читает содержимое файла 1;
- находит в файле заданную вторым параметром процедуры 1 строку;
- записывает найденную строку в файл 2;
- посылает процедуре 1 сигнал 2, после чего завершает свою работу.

Третья процедура:

- проверяет циклически, завершились ли процедуры 1 и 2 и, в случае их завершения,
- записывает в файл 3 некоторое сообщение, после чего завершает свою работу.

Вариант 109

Написать 2 shell-процедуры

Первая процедура:

- при первом получении сигнала <ctrl+c> выдает на терминал сообщение «Доброе утро»;
- при втором получении сигнала <ctrl+c> выдает на терминал сообщение «Добрый день»;
- при третьем получении сигнала <ctrl+c> выдает на терминал сообщение «Добрый вечер»;
- при получении сигнала <ctrl+\> запускает n раз (n передается в процедуру в качестве параметра) в фоновом режиме процедуру 2 и на этом завершает свою работу.

Вторая процедура:

- создает в текущем каталоге подкаталог C, делает его текущим и на этом завершает свою работу.

Вариант 110

Написать 2 shell-процедуры

Первая процедура:

- при первом получении сигнала <ctrl+c> выдает на терминал сообщение «Введите n»;
- вводит число n;
- при втором получении сигнала <ctrl+c> выдает на терминал сообщение «Введите строку»;
- вводит запрошенную строку;
- при получении сигнала <ctrl+\> запускает n раз (n передается в процедуру в качестве параметра) в фоновом режиме процедуру 2 и на этом завершает свою работу.

Вторая процедура:

- создает в текущем каталоге подкаталог C и делает его текущим;
- создает в новом текущем каталоге новый файл F;
- записывает в файл F введенную процедурой 1 строку и на этом заканчивает свою работу.

Приложение 1. Основы работы с редактором VI

Для редактирования текстовых документов в рамках практикума используется консольный редактор VI. Для запуска редактора наберите следующую команду:

$ vi имя_файла

Данная команда откроет файл *имя_файла* в VI. Если файл с именем *имя_файла* отсутствует, то он будет создан.

Пользователь может взаимодействовать с редактором VI в двух режимах: режиме ввода команд и режиме ввода текста. Нажмите клавишу *i* для перехода в режим ввода текста. Для возвращения в режим команд используйте клавишу *ESC*.

Основные команды, используемые в редакторе VI:

<Esc>:wq! - выход с сохранением файла
<Esc>:q! - выход без сохранения файла

Ввод текста:

i/I (insert) - вставка текста в начало текущей строки
a/A (append) - набор текста в конец текущей строки
o/O (open) - вставить пустую строку после/перед текущей

Команды удаления текста (в буфер):

x - удаление текущего символа
[#]dw - удаление # текущих слов
[#]dd -удаление # текущих строк
dG - удаление всех строк от текущей до последней
d$ - удаление конца строки от текущей позиции
d^ - удаление начала строки до текущей позиции.

Команды отмены произведенных изменений в текущей строке:

u - отмена (undo) последнего изменения
U - отмена всех изменений.

Команды копирования указанного в команде объекта в буфер:

[#]yw - копирование (yank) текущего слова
[#]yy - копирование текущей строки
yG - копирование строк от текущей до конца файла
y$ - копирование части строки от курсора до конца строки
y^ - копирование части строки от курсора до начала строки

Команды вставки буфера в текст:

p/P - после/перед курсором

Команды изменения текста:

r<символ> - заменяет (change) текущий символ на указанный
c<объек> - заменяет указанный объект на текст, с клавиатуры - <Esc>:
#cw -изменение текущего слова
#cc - всей текущей строки
cG - всех строк файла от текущей до последней
cS -части строки от курсора до конца строки
c^ - части строки от курсора до начала строки.

Команды поиска строки файла по фрагменту её текста:

/<текст> - от текущей строки до конца файла с переходом на начало
?<текст> - от текущей строки до начала файла и по всему файлу

n - поиск следующей строки в файле, аналогичной найденной

N - поиск предыдущего вхождения в файл заданной строки.

Префиксные команды:

:w файл - запись текущего буфера в файл

:m,nw файл - запись строк с m-ой по n-ую в файл

:e! - отмена всех изменений в буфере с перезагрузкой в него файла с диска

:e файл - загрузка файла в буфер с замещением старого содержимого

:r файл - добавление содержимого файла после текущего положения курсора

:set опция - настройка редактора (см. полное описание vi).