ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

МОСКОВСКИЙ ИНЖЕНЕРНО-ФИЗИЧЕСКИЙ ИНСТИТУТ
(ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ)

С.В. Ктитров Н.В. Овсянникова

КОМАНДНЫЙ ЯЗЫК ОС UNIX

Лабораторный практикум

Рекомендовано УМО «Ядерные физика и технологии»
в качестве учебного пособия
для студентов высших учебных заведений

Москва 2007

УДК 004.451.9(075)
ББК 32.973-018.2я7
К87

Ктитров С.В., Овсянникова Н.В. Командный язык ОС UNIX. Лабораторный практикум. М.: МИФИ. 2007. – 60 с.

Пособие содержит задачи и упражнения для практического освоения приемов и методов работы в операционной системе UNIX в качестве пользователя текстового терминала. Задачи охватывают как основные пользовательские команды, редактор vi, так и более сложные вопросы, в частности, резервное копирование. Основное внимание уделяется выработке навыков программирования с использованием командного языка, перехвату и обработке сигналов, управлению заданиями, защите файлов.

Пособие предназначено для студентов, изучающих курс «Операционная система UNIX», и может служить основой как лабораторного практикума, так и самостоятельной работы.

Пособие подготовлено в рамках Инновационной образовательной программы.

Рецензент канд. техн. наук, доц. А.Б. Вавренюк

ISBN 978-5-7262-0851-0

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Операционная система UNIX с самого начала создавалась как инструментальная среда, которая предлагает развитые средства для быстрого и простого построения программных инструментов при решении самых разнообразных задач путём совместной работы простых специализированных программ [3-5,10,11,14]. Такой подход потребовал создания чрезвычайно развитого командного языка, по мощности предоставляемых средств сопоставимого с языками программирования. Трудно назвать задачу, которую нельзя было бы решить только его средствами. Под командным языком здесь подразумевается язык интерпретатора команд (*shell*) и стандартные программы, которые пользователь обычно считает элементами языка системы, т. е. языка, на котором с системой общается пользователь, а не программист.

Владение командным языком операционной системы – одна из важнейших составляющих компьютерной грамотности. Для освоения большинства предоставляемых интерпретатором команд операционной системы *UNIX* возможностей потребуется много усилий. Частично облегчить эту задачу, указав наиболее интересные и важные особенности системы, призван данный лабораторный практикум.

Практикум является сборником задач и упражнений разного уровня сложности, начиная с задач на использование основных команд, которые позволят учащемуся освоиться в системе, понять основные принципы ее функционирования, заканчивая достаточно сложными задачами, требующими написания нескольких взаимодействующих сценариев. Задачи разбиты на группы в соответствии с изучаемой темой. Уровень сложности задач постепенно повышается, и, как правило, задачи каждой темы требуют освоения материала предшествующих разделов.

В сборник включены задачи по следующим темам: основные команды пользователя, взаимодействие пользователей между собой в многопользовательской среде, обмен сообщениями и файлами, навигация по файловой системе и защита файлов и каталогов, конвейеры и перенаправление ввода/вывода, управление заданиями, создание индивидуальной рабочей среды, написание сценариев (*shell*-программ), обработка и использование сигналов, поиск файлов и создание архивов. Полученные при выполнении упраж-

нений навыки позволяют не только существенно облегчить использование *UNIX* для индивидуального пользователя, пояснить и проиллюстрировать основные концепции операционной системы, но и выработать навыки, которые могут оказаться полезными администраторам системы.

Все представленные упражнения могут быть выполнены на любой *UNIX*-подобной системе. Возможно, для некоторых систем решения будут слегка различаться вследствие особенностей реализации отдельных команд. Исключение составляют несколько задач из раздела 7, использующих списки управления доступом, которые могут не поддерживаться некоторыми файловыми системами. Предполагается, что для решения задач используется *Korn shell* или *bash*, который является основным в ОС Linux и доступен в других ОС, а для редактирования файлов редактор *vi* или его дальнейшее развитие – *vim*. Альтернативный *C shell* читатель может освоить, например, по [7].

Наиболее трудные задачи снабжены указаниями к решению, однако следует понимать, что для ряда задач (и реализаций операционных систем) решение неоднозначно, и верное решение может отличаться от предложенного.

В имеющейся на русском языке литературе командный язык описан довольно подробно [1,8,9,12], поэтому в практикуме приводятся лишь основные сведения о командах, носящие характер указаний. К сожалению, некоторым чрезвычайно полезным его средствам и «тонким особенностям» обычно уделяется незаслуженно мало внимания или они вовсе остаются «за кадром». Некоторые из них описаны в настоящем практикуме. Авторы надеются, что приводимый материал облегчит решение некоторых задач и приоткроет некоторые грани богатого языка *UNIX*.

В заключение авторы хотели бы высказать надежду, что читатель не станет ограничиваться лишь приведенными задачами и, проявив свою фантазию, сформирует для себя удобный набор инструментов, сделав свою систему комфортной для работы, как о том мечтали авторы *UNIX*.

Разделы 2,4,5,6,7 написаны С.В. Ктитровым, разделы 1,3,8 – Н.В. Овсянниковой.

1. ОСНОВНЫЕ КОМАНДЫ

Для выполнения заданий этого раздела Вам потребуется знание следующих команд:

- *who* – вывод информации об активных пользователях;
- *whoami, who am i* – вывод информации о пользователе терминала;
- *uname* – вывод информации о версии операционной системы;
- *echo* – вывод сообщений на терминал;
- *banner* – вывод сообщений на терминал прописными буквами;
- *man* – вызов оперативной справочной системы;
- *date* – вывод текущей даты;
- *cal* – календарь;
- *write* – передача сообщений на терминал другого пользователя;
- *wall* – передача сообщений на все терминалы;
- *mesg* – разрешение/запрет вывода сообщений от других пользователей;
- *mail* – отправка/получение почты;
- *news* – знакомство с новостями системы;
- *ps* – печать списка запущенных программ;
- *clear* – очистка экрана;
- *login* – регистрация в системе;
- *logout* – выход из системы;
- *ksh* – запуск Korn shell;
- *bash* – запуск *Bourne-Again Shell*;
- *exit* – завершение текущего *shell*.

Обратите внимание

Если команда читает текст с терминала, как, например, команды передачи сообщений, конец текста следует обозначить символом ^D. Тем самым программе передается символ «конца файла».

Система новостей в Linux обычно не используется, но присутствует по умолчанию в Solaris, HP-UX.

Справочная система man может содержать одноименные статьи в разных разделах. Например, umask из раздела 1 – это команда, из раздела 2 – системный вызов. Если раздел не указан, по умолчанию отображается, как правило, статья раздела 1. Для доступа к разделу 2 следует ввести:

```
man 2 umask
```

или

```
man -S 2 umask
```

В некоторых системах поддерживаются оба варианта, в некоторых какой-либо один.

Строка-аргумент команды echo может содержать управляющие символы, обозначения которых аналогичны используемым в программах на языке Си. Это позволяет передавать одной командой форматированный текст. Такой текст следует заключать в двойные кавычки и указывать опцию *-e*. Ознакомление с другими опциями оставлено в качестве упражнения (посмотрите *man*).

Пример.

С помощью команды echo вывести в столбик цифры от 0 до 9.

Решение.

```
echo -e "0\n1\n2\n3\n4\n5\n6\n7\n8\n9\n"
```

Упражнения

1.1. Ответьте на следующие вопросы:

Сколько пользователей работает в системе в настоящий момент?

Кто работает в системе ?

Через какой терминал Вы вошли в систему ?

1.2. Какие из предложенных ниже команд приведут в выдаче приветственного сообщения на экран? Какие нет? Почему?

```
echo
echo hello
echohello
echo HelLO
Echo hello
echo HELLO, WORLD
banner
banner hello
BANNER HELLO
```

1.3. Выведите сообщение из нескольких строк с помощью команд *echo* и *banner*.

1.4. Прочитайте статью справочной системы о пользовании справочной системой.

1.5. Прочитайте статью справочной системы о команде *uname*. Из какого раздела справочника Вы прочитали статью? Как прочитать статью об этой команде из раздела 2?

1.6. Определите имя машины, название и версию операционной системы, с которой Вы работаете. Каков аппаратный тип системы?

1.7. Выведите дату в форматах *dd-mm-yy, mm-dd-yy hh:mm:ss.*

1.8. Выведите дату в две строки: на первой – день, месяц, год, на второй – текущее время, снабдив вывод комментарием.

1.9. Выполните задание 1.8 так, чтобы на экране не было посторонней информации, а только результат выполнения задания.

1.10. Измените текущее значение даты. Какова реакция системы? Почему?

1.11. Выведите календарь на текущий месяц, на другой месяц текущего года.

1.12. Выведите календарь на будущий год. В каком столбце отображаются воскресенья? Как сделать, чтобы неделя начиналась с понедельника? С воскресенья?

1.13. Определите порядковый номер текущего дня с начала года.

1.14. Напечатайте календари на 2 года сразу одной командной строкой.

1.15. Напечатайте календарь на будущий год с «линиями отреза» до и после календаря, отделив их тремя пустыми строками.

1.16. Используя команду *write*, пошлите сообщение на консоль.

1.17. Пошлите сообщение на соседний терминал.

1.18. Пошлите сообщение на все терминалы одновременно.

1.19. Используя команду *mesg*, определите, разрешены ли сообщения на Ваш терминал. Запретите сообщения. Какова будет реакция системы, если кто-нибудь попытается передать Вам сообщение?

1.20. Прочитайте свою почту в порядке поступления. Пошлите письмо товарищу. Прочитайте почту в обратном порядке. Перешлите последнее сообщение на консоль, первое сообщение сохраните в файле *mymail*, а остальные сообщения уничтожьте.

1.21. Прочитайте новости системы. Как прочитать все новости еще раз?

1.22. Запустите новый экземпляр интерпретатора команд, не выходя из системы. Запустите другой интерпретатор команд.

1.23. Завершите работу всех интерпретаторов из задания 1.22. Сколько команд потребовалось выполнить?

1.24. Определите, сколько и каких интерпретаторов запущено на данном терминале.

1.25. Завершите сеанс. Как понять, что Вы вышли из системы?

2. КОМАНДЫ РАБОТЫ С ФАЙЛАМИ И КАТАЛОГАМИ

Для выполнения приведенных ниже заданий рекомендуется использовать следующие:

- *pwd* – вывод абсолютного маршрутного имени текущего рабочего каталога;
- *cd* – изменение рабочего каталога;
- *ls* – вывод информации о содержимом каталога;
- *mkdir* – создание каталога;
- *rmdir* – удаление каталога;
- *touch* – обновление временной метки файла;
- *cp* – копирование файлов;
- *mv* – перемещение или переименование файла;
- *cat* – объединение и вывод на экран содержимого файлов;
- *more* – постраничный просмотр содержимого файла;
- *rm* – удаление файла;
- *ln* – создание ссылки на файл;
- *alias* – создание псевдонима.

Для выполнения ряда заданий потребуется знание шаблонов генерации имен файлов:

? (вопросительный знак) соответствует любому одному символу, кроме первой точки;

[ ] *(квадратные скобки)* определяют группу символов (выбирается один символ из группы);

- *(знак «минус»)* определяет диапазон допустимых символов;

! (восклицательный знак) отвергает следующую за ним группу символов;

* *(символ «звездочка»)* соответствует любому количеству символов, кроме первой точки.

Обратите внимание

Именно интерпретатор команд заменяет метасимволы *, ?, [ ] на соответствующие им имена файлов. Программа вместо аргументов, содержащих метасимволы, получит сформированный интерпретатором список – результат подстановки, а число аргументов может измениться. Вот почему от команды

```
cp  *.cpp  *.exe
```

не следует ожидать попарной подстановки, привычной на персональных ЭВМ, где она реализуется средствами самой программы. С другой стороны, подход UNIX позволяет единообразно применять символы экранирования *", ‘* и, тем самым, более гибко управлять поведением программы.

Пример 1.

Вывести на экран содержимое всех файлов, имеющих в имени символ `e`.

```
cat  *e*
```

Файлы, имя которых начинается с символа . (точка), не подпадают под указанный шаблон, поэтому если требуется отобразить и такие файлы, команду следует дополнить:

```
cat  *e* .*e*
```

Пример 2.

Вывести в столбик список файлов, в именах которых второй символ – пробел.

```
ls -1 ?\ *
```

Пример 3.

Переместить файл *listing* из каталога *work* в каталог *Development_And_Research*. Других каталогов и файлов, имя которых начинается на букву *D* в текущем каталоге нет.

```
mv work/listing D*/
```

Пример 4.

Создать псевдоним *sl*, который создает символическую ссылку на заданный файл.

```
alias sl="ln -s"
```

Упражнения

2.1. Используйте команды *cat* и *more* для вывода на экран содержимого каталога. Что произошло? Почему? Какую команду следует использовать для вывода содержимого каталога?

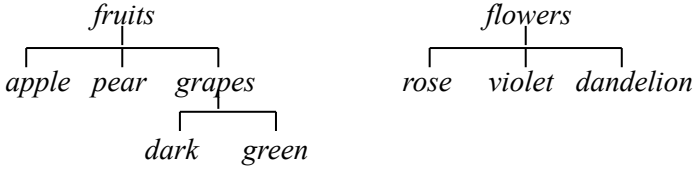
2.2. Посмотрите содержимое Вашего домашнего каталога.

2.3. Определите имя своего домашнего каталога и его родительского каталога.

2.4. Определите полный путь до Вашего домашнего каталога.

2.5. Постройте поддерево каталогов, начиная с каталога */home* (или */export/home* в ОС Solaris) с помощью команд *cd, ls* и *pwd.*

2.6. Создайте в своем домашнем каталоге подкаталоги вида:



2.7. Находясь в своем домашнем каталоге, создайте следующие подкаталоги, используя одну командную строку:

A/B/C;
A/B;
A/B/C/D;
A/E.

2.8. Находясь в своем домашнем каталоге, удалите все подкаталоги каталога *A*.

2.9. Создайте пустой файл с помощью команды *touch.*

2.10. Находясь в домашнем каталоге, создайте файл *macintosh* в имеющемся каталоге *apple* и несколько файлов в каталогах *dark* и *green*.

2.11. Войдите в каталог *flowers*. Находясь в каталоге *flowers,* скопируйте все подкаталоги *fruits* вместе с находящимися в них файлами в специально созданный каталог *basket.*

2.12. Находясь в каталоге *flowers*, удалите каталог *fruits.*

2.13. Прочитайте файл *.profile* (или *.bash_profile*) с помощью команд *cat* и *more.*

2.14. Создайте копию файла из задачи 2.13, назвав её *myfile* или создайте новый файл с таким именем.

2.15. Просмотрите содержимое созданного в задаче 2.14 файла *myfile*. Скопируйте файл *myfile* в файл *mycopy.* Просмотрите содержимое обоих файлов.

2.16. Удалите файл *myfile.* Что произошло с файлом *mycopy*?

2.17. Переместите файл *mycopy* в каталог *flowers.*

2.18. Находясь в домашнем каталоге, создайте ссылку *mylink* на файл *mycopy*, находящийся в каталоге *flowers*. Просмотрите файл-ссылку.

2.19. С использованием ссылки *mylink* восстановите файл *myfile.*

2.20. Добавьте строку к файлу *mylink.* Какие из файлов *mylink, mycopy, myfile* изменились? Почему?

2.21. С использованием какой команды можно определить число ссылок? Определите число ссылок для файлов *mylink, mycopy, myfile.*

2.22. Удалите файл *mycopy.* Что произошло с файлом-ссылкой? Определите число ссылок для файлов *mylink, myfile.*

2.23. Создайте псевдоним *dir*, распечатывающий содержимое текущего каталога в расширенном формате.

2.24. Завершите сеанс работы и вновь зарегистрируйтесь. Работает ли псевдоним *dir*? Что следует сделать, чтобы псевдоним «не терялся» между сеансами?

2.25. Создайте псевдоним *del*, удаляющий указанный каталог со всеми содержащимися в нем подкаталогами и файлами, с подтверждением каждого удаления.

2.26. Создайте псевдоним *point*, распечатывающий список находящихся в рабочем каталоге файлов, имена которых начинаются с точки.

2.27. Создайте набор псевдонимов, реализующих мусорную корзину, которые позволяют удалять файл в корзину, восстанавливать файл из корзины, очищать её. Как сделать, чтобы корзина не была бы видна пользователю при обычной работе.

2.28. Создайте псевдоним *joke*, сообщающий, что все файлы в текущем каталоге инфицированы и будут уничтожены, и выдающий список этих файлов.

2.29. Модифицируйте псевдоним из упражнения 2.28 так, чтобы он стал «одноразовым», т.е. допускал только однократное использование.

2.30. Используя команду *touch*, создайте файлы в новом каталоге с такими именами, чтобы одновременно:

шаблону *a** соответствовали 5 файлов;
шаблону **a* соответствовали 4 файла;
шаблону *??.?* соответствовали 3 файла;
шаблону **aa** соответствовали 2 файла;
шаблону ??? соответствовал 1 файл.

2.31. Какую команду следует ввести, чтобы сделать следующее:

а) вывести все имена файлов, начинающиеся с точки;

б) вывести все имена файлов, оканчивающиеся на *.txt*;

в) вывести все имена файлов, содержащие слово *my*;

г) вывести все имена файлов, начинающиеся на *my* и содержащие ровно 5 символов;

д) вывести все имена файлов, оканчивающиеся на *.c, .f* или *.p*.

2.32. Создайте файл с именем *. Удалите только этот файл.

2.33. Создайте файл, в имени которого есть символ «пробел». Как удалить такой файл?

2.34. Создайте несколько файлов, имена которых состоят только из пробелов. Удалите один из таких файлов с заданным числом пробелов в имени. Удалите все такие файлы.

2.35. Создайте символическую ссылку на файл *document*. Посмотрите содержимое файла, используя ссылку.

2.36. Создайте ещё одну ссылку на файл из задачи 2.35. Посмотрите количество ссылок на файл командой ls. Объясните результат.

2.37. Определите, является ли файл ссылкой. Какой?

2.38. Удалите файл из задачи 2.35. Обратитесь к нему по ссылке. Объясните результат.

2.39. Создайте в каталоге *apple* непустой файл *green*. Создайте ссылку *hlgreen* и символическую ссылку *slgreen*. Просмотрите содержимое этих файлов, их тип и число ссылок. Удалите файл *green* и вновь просмотрите содержимое и свойства файлов-ссылок. Создайте новый файл *green* с другим содержимым и сравните тип, число ссылок и содержимое всех файлов.

2.40. Создайте ссылку *hlapple* и символическую ссылку *slapple* на каталог *apple*. Сравните содержимое каталогов *apple*, *hlapple* и *slapple*. Удалите каталог *apple* и вновь создайте, разместив в нем файл *red*. Сравните содержимое каталогов apple, *hlapple* и *slapple*. Объясните результат.

3. ТЕКСТОВЫЙ РЕДАКТОР *vi*

О редакторе *vi* хотелось бы сказать особо. Обычно указывается, что его изучение полезно, потому что это единственный редактор, присутствующий в любой системе *UNIX*. Однако связь его с системой глубже. Команда *set –o vi*, задающая режим интерпретатора команд, позволяет редактировать командную строку с использованием команд *vi* (точнее, строкового редактора *ex*, лежащего в его основе). Своеобразие его команд вызвано стремлением обеспечить его работу на любом оборудовании. Возможно, в повседневной работе эффективнее использовать какой-либо другой редактор, но уметь работать с *vi* необходимо. Дальнейшее развитие редактора носит название *vim (Vi IMproved)*. Этот редактор наряду с традиционными командами и стилем работы *vi* поддерживает стиль, более привычный пользователям современных систем. Вполне вероятно, что командой *vi* будет запущен *vim*. *Авторы настоятельно рекомедуют освоить приведенные ниже команды и выработать навыки их использования, не поддаваясь соблазну воспользоваться "благами цивилизации"*.

Основные команды текстового редактора:

ESC	переход в командный режим;
h	курсор влево;
l	курсор вправо;
k	курсор вверх;
j	курсор вниз;
0	курсор к началу строки;
$	курсор к концу строки;
G	курсор к началу файла;
1G	курсор к концу файла;
w	к следующему слову;
b	к предыдущему слову;
e	к концу слова;
^	к первому непустому символу строки;
^D	пролистать вниз;
^U	пролистать вверх;
^E	переместить окно на одну строку вниз;
^Y	переместить окно на одну строку вверх;
^L	очистить и перерисовать окно;

mx пометить текущию позицию буквой *x*;
'x переместить курсор в позицию, помеченную *x*;
H переместить курсор на верхнюю строку на экране;
L переместить курсор на нихнюю строку на экране;
M переместить курсор на среднюю строку на экране;
+ переместить курсор к первому непустому символу следующей строки;
- переместить курсор к первому непустому символу предыдущей строки;
i режим вставки;
I режим вставки перед первым непустым символом;
a режим добавления текста;
A режим добавления текста в конец строки;
x удалить символ;
X удалить символ перед курсором;
dd удалить строку;
3*dd* удалить 3 строки;
dw удалить слово;
D удалить от курсора до конца строки;
dG удалить от курсора до конца файла;
< сдвинуть строку влево;
> сдвинуть строку вправо;
yy копировить строку в буфер;
3yy копировить 3 строки в буфер;
3yl копировить 3 символа в буфер;
p вставить после курсора содержимое буфера;
P вставить перед курсором содержимое буфера;
u отменить действие;
U восстановить строку;
/word найти слово (поиск вперед);
?*word* найти слово (поиск назад);
n продолжить поиск по команде / или ?;
N продолжить поиск по команде / или ? в противоположном направлении;
% найти соответствующие ( ) или { };
ZZ выйти с сохранением изменений;
:q выйти из редактора;
:q! выйти без сохранения изменений;

:w	записать изменения;
:w file	записать в заданный файл;
:w! file	записать в заданный существующий файл;
:e file	редактировать файл *file*;
^G	показать текущие файл и строку;
:sh	запустить *shell;*
:!cmd	выполнить команду.

Упражнения

3.1. В Вашем личном каталоге находится файл *vi.txt*, текст которого приведен ниже. Откройте сеанс редактирования для этого файла и выполните инструкции, указанные в файле:

```
   Enter your name here ->

   Change the following to your favourite colour
->pink
   Change the following to your favourite flower
->violet
   Change the following to your favourite book
->"Lobo"

   Correct mistakes in the following lines:
   There awe meny misteiks in thiss line
   Ther  are less mistakes in this line
   Threr are onli thhree mistakes in this line

   Delete any occurrence of the word "very" in the
next line:
   Very many very rich people do not very like to
spend very much money

   Change any occurrence of the word "like" to "hate"
in the next lines:
   Most people like porridge but very few like meat.
   Girls like mice and mice like cats.

   Arrange the lines in the right order and change
every "A" to "a":

   1.In the town where I wAs born
   7.And we lived beneAth the wAves,
   2.lived A mAn who sAiled the seA,
   8.In our yellow submArine.
```

```
   3.And he told us of his life,
   4.In the lAnd of submArines.
   6.till we found the seA of green,
   5.So we sAiled on to the sun,

   Now copy 8 previous lines to the file subma-
rine.txt.

   Save the changes you have made.

   Match the book and its author:

   "Alice in Wonderland"Daniel Defoe
   "Robinson Crusoe"    Lev Tolstoy
   "Anna Karenina"    Lewis Carroll
   "Ivanhoe"     Herbert Wells
   "The Time Machine" Walter Scott

   Remove the duplicate lines:

   Is there anybody going to listen to my story
   Is there anybody going to listen to my story
   all about the girl who came to stay?
   She's the kind of girl you want so much
   it makes you sorry
   it makes you sorry
   it makes you sorry
   still you don't regret a single day.
   Ah girl, girl.

   Remove all the blank lines.

   Enter a blank line before the 1st and the 10th
lines.

   Save the changes you have made and quit.
```

3.2. Запустите *korn-shell*. Переведите его в режим редактора *vi*. Выполните несколько команд. Листая список, выберите предыдущую команду, подредактируйте её и выполните. Используйте команды *vi.*

3.3. Найдите файл с историей команд в своём домашнем каталоге и подредактируйте его, используя *vi.* Проверьте историю команд.

4. ПЕРЕНАПРАВЛЕНИЕ ВВОДА/ВЫВОДА. КОНВЕЙЕРЫ

Перенаправление ввода/вывода и конвейеры лежат в основе механизма связывания программ ОС UNIX, что позволяет говорить о ней как об инструментальной среде, в которой пользователь, объединяя простые программы, создает средства для решения сложных задач.

Перенаправления стандартного вывода

программа > файл

ввода

программа < файл

и стандартного вывода сообщений об ошибках

программа 2> файл

получили распространение и вне ОС UNIX. Например, стандартный интерпретатор команд Microsoft Windows реализует указанные команды, так же, как и конвейеры.

Перенаправление потоков ввода/вывода не ограничивается в UNIX широко известными командами < и >. Стандартный ввод может быть закрыт *<&–* (аналогично, закрытие стандартного вывода *>&–*) или перенаправлен в файл с дескриптором *N*: *<&N* (соответственно, для потока вывода *>&N*). Более того, численный описатель файла может появиться перед символом < или >, что позволяет осуществлять сложные перенаправления, интерпретируемые слева направо (дескрипторы 1 и 2 связываются с файлом *file*):

```
2>file 1>&2
```

Файл может быть открыт для чтения и записи одновременно с использованием <>.

В данный раздел включены задачи на следующие команды:

- < – перенаправление ввода;
- >, >> – перенаправление вывода;
- *2>, 2>>* – перенаправление сообщений об ошибках;
- | – конвейер;
- *tee* – перехват результатов конвейера;
- *cut* – выделение полей из строки;
- *grep* – поиск по шаблону;

- *pr* – вывод файла на стандартный вывод в заданном формате;
- *sort* – сортировка;
- *head* – вывод первых строк файла;
- *tail* – вывод последних строк файла;
- *wc* – подсчет количества символов, слов и строк;
- *tr* – преобразование символов.

Пример 1.

Записать слово word одновременно в два файла.

```
echo word | tee file1 >file2
```

Пример 2.

Напечатать на экране четвертую строку файла myfile (предполагается, что в файле не менее четырех строк).

```
head -n 1 myfile | tail -n 1
```

Пример 3.

Определить последнее воскресенье текущего месяца и выдать соответствующее пояснение.

```
echo -n "Last Sunday is "; cal -s | tail -n 2 \
| tr -b1-3
```

Решение будет верным, если воскресенья указаны в первой колонке, для чего указывается опция *-s*. В некоторых системах данная опция не поддерживается, однако, как правило, воскресенье начинает неделю. Если это не так, следует установить переменную *LC_TIME=C.*

Упражнения

4.1. Выведите текущую дату на экран большими буквами с помощью команд *date* и *banner.*

4.2. Создайте в домашнем каталоге текстовый файл *myfile* из нескольких строк с помощью команды *cat.* Создайте текстовый файл *MyFile,* записав в него такие же строки. Сколько файлов у Вас получилось? Почему?

4.3. Добавьте к файлу из задачи 4.2 две строки. Какую команду Вы использовали? Какой ещё командой это можно было сделать?

4.4. Перейдите в домашний каталог. Перенаправьте вывод команды *ls* в файл *spisok*. Просмотрите файл *spisok*. Сравните с выводом на терминал. Что произошло? В чем отличие? Как исправить?

4.5. С использованием команды *cat* удвойте содержимое файла, приписав исходное содержимое к концу этого же файла.

4.6. Просмотрите содержимое файла, полученного в предыдущей задаче. Как следует поступить, если вывод файла командой *cat* не умещается на экране?

4.7. Задайте псевдоним, создающий текстовый файл, на основе команды *cat*. Пользователь получает приглашение на ввод и указывает имя результирующего файла.

4.8. Перейдите в каталог *flowers*. Добавьте в файл *spisok* список содержимого каталога *flowers.*

4.9. Создайте два непустых файла *f1* и *f2* с помощью *cat*. Создайте файл *f3*, содержащий информацию из файлов *f1* и *f2*. Просмотрите файл *f3*.

4.10. Попытайтесь прочитать с помощью команды *cat* несуществующий файл. Какова реакция системы? Сделайте то же самое, перенаправив сообщения об ошибках в файл *myerror.* Что Вы видите на экране? Просмотрите файл *myerror.*

4.11. Создайте конвейер для получения списка только числовых идентификаторов пользователей, работающих в системе.

4.12. Создайте конвейер для получения списка только имен и прав доступа к файлам, которые в данный момент находятся в Вашем рабочем каталоге.

4.13. Измените построенный конвейер так, чтобы список сохранялся в файле *spisok* Вашего домашнего каталога, а на экран выводилось только число файлов в списке.

4.14. Распечатайте файл **.***profile* столбцами по 20 символов, разделяя на страницы по 20 строк, и сохраните результат в файле *example.*

4.15. Выведите на экран содержимое файла */etc/passwd*, упорядоченное по полю с именем пользователя.

4.16. Выведите на экран содержимое файла */etc/passwd*, упорядоченное по третьему полю.

4.17. Выведите на экран все строки файла */etc/passwd*, содержащие слово *st*. Сохраните результат в файле *st_users*. Подсчитайте количество строк в файле *st_users.*

4.18. Выведите на экран все строки файла */etc/passwd*, не содержащие слово *st.* Подсчитайте количество таких строк.

4.19. Создайте файл с именем, содержащим две точки, окруженные другими символами (вида *prefix.infix.suffix*). Выведите на экран полное имя файла, часть имени файла без суффикса (т.е. часть имени до правой точки), имя файла без префикса (т.е. часть имени файла справа от первой точки) и среднюю часть имени файла, заключенную между точками.

4.20. Создайте конвейер, который будет подсчитывать количество пользователей, работающих в системе.

4.21. Измените построенный в предыдущей задаче конвейер так, чтобы он сохранял список пользователей в файле *users.*

4.22. Создайте псевдоним *loggedon*, который будет выводить на экран упорядоченный в алфавитном порядке список имен работающих в системе пользователей.

4.23. Создайте конвейер, который выводил бы на экран упорядоченный по алфавиту список зарегистрированных пользователей, содержащий только имена пользователей, их идентификационные номера и имена домашних каталогов.

4.24. Выделите из вывода команды *date* месяц, день и год.

4.25. Постройте конвейер, выводящий на экран строки файла */etc/passwd* с пятой по десятую, упорядоченные по третьему полю.

4.26. Используя команду *cat*, одновременно выведите на экран содержимое трех файлов и скопируйте его в четвертый файл.

4.27. Одной командой *cat* запишите в файл содержимое двух различных файлов, разделив их текстом, введенным с клавиатуры.

4.28. В одной командной строке сохраните командой *cat* содержимое трех файлов одновременно в двух файлах в разных каталогах.

4.29. Запишите вывод команды *cat* одновременно в два файла, в один файл – отсортированный вывод, в другой файл – неотсортированный.

4.30. Замените в одном заданном файле все строчные буквы на прописные, в другом – все прописные на строчные.

4.31. Исключите из файла все повторяющиеся пробелы.

4.32. Замените в указанном файле все символы табуляции на символ «пробел», а все символы *$* – на *@*.

4.33. Напечатайте отсортированный список пользователей, путевое имя домашнего каталога которых содержит каталог *home*.

4.34. Модифицируйте решение задачи 4.32 так, чтобы дополнительно сообщалось общее количество пользователей и число найденных пользователей.

5. ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ КОМАНДНОГО ЯЗЫКА

Функции и сценарии. Под сценарием (*script*) подразумевается набор команд (встроенных, т. е. понимаемых самим *shell*, например, *if*, и внешних, т. е. программ и других сценариев), которые записаны в файле (командный файл). Для запуска сценария на выполнение файл сценария требуется пометить как выполняемый или указать имя файла в качестве аргумента программы, реализующей *shell*, такой как *sh* или *ksh*. В любом случае это приведет к порождению нового процесса. Это не всегда желательно, причем не только по соображениям производительности. Например, дочерний процесс не может изменить переменные родительского процесса, хотя такой обмен данными часто бывает полезен. Для запуска сценария в текущем интерпретаторе имя файла сценария следует предварить командой *eval*. Команда *exec* заменяет программой – своим аргументом – текущий процесс. Например, сценарий

```
echo line 1
eval pwd
echo line 2
exec pwd
echo line 3
```

приведет к следующему выводу (предполагается, что для его запуска создается порожденный *shell*, результат его запуска командой *eval* читателю предлагается получить экспериментально и объяснить самостоятельно):

```
line 1
/export/home/user1
line 2
/export/home/user1
```

Функции, могут быть заданы не только внутри *shell*-программы, но и непосредственно в командной строке следующим образом:

function имя {команды;}

или

имя() {команды;}.

Особенностью функций является их хранение в памяти наряду с переменными окружения, что позволяет не только ускорить их выполнение, но и не приводит к порождению нового процесса для исполнения. Для выхода из функции можно использовать return, с

необязательным аргументом, задающим код завершения. В остальном функции ведут себя как *shell*-программы. Для удаления функций, которые хранятся не в файле, требуется использовать команду *unset –f*. Порожденный *shell* может быть создан явно с использованием команды *command*.

Обработка значений переменных. Как и любой интерпретатор команд, *shell UNIX* обеспечивает возможность работы с переменными, имеющими значение – строку символов. Наличие условных подстановок позволяет манипулировать значениями переменных кратко, изящно и выразительно. Подстановка значения переменной производится конструкцией

```
${var}
```

Некоторые условные подстановки:

`${var:-word}`	подставляется *word*, только если *var* не присвоено значение, иначе подставляется *var*;
`${#var}`	если *var* – * или @, подставляется число позиционных параметров, иначе длина значения *var* в байтах;
`${var%word}`	подставляется значение *var* после удаления из его конечной части наименьшего фрагмента, соответствующего шаблону *word*;
`${var%%word}`	подставляется значение *var* после удаления из его конечной части наибольшего фрагмента, соответствующего шаблону *word;*
`${var#word}`	подставляется значение *var* после удаления из его начальной части наименьшего фрагмента, соответствующего шаблону *word;*
`${var##word}`	подставляется значение *var* после удаления из его начальной части наибольшего фрагмента, соответствующего шаблону *word.*

Например,

```
x=file.c++
echo ${x%.c++}.out
x.out
echo ${#x} symbols
8 symbols
```

Переменные могут быть помечены как неизменяемые (команда *readonly*). Командой *typeset* можно задать регистр их символов.

При решении задач этого раздела используйте следующие конструкции командного языка:

- подстановка переменных (*$имя_переменной*);
- подстановка команд (*$(команда)* или *`команда`*);
- чтение и вывод значений переменных (команды *read, echo*);
- передача аргументов *shell*-программе через переменные среды и через аргументы командной строки;
- оценивание значения выражения (команда *test*);
- вычисление арифметических выражений (команды *expr* и *let*);
- функции (*function*);
- условные операторы и операторы ветвления (*||, &&, if-then, if-then-else, case*);
- операторы цикла (*for, while, until*);
- формирование кода возврата в *shell*-процедуре и *shell*-функции (*exit, return*);
- команды манипулирования значениями переменных (*${%}, ${%%}, ${#}, ${##}, $#*).

В задачах под термином «расширение имени файла» подразумевается конечная часть имени файла после символа «точка».

Пример 1.

Найдите в текущем году все месяцы, в которых заданный день недели приходится на заданное число.

```
export LC_TIME=C
nday=$1
day=$(($2 * 3 ))
ccol=$day-$(($day+3))
year=$(date | cut -d' ' -f6)
for month in 1 2 3 4 5 6 7 8 9 10 11 12
do
  lst=$(cal $month $year|tail -n 6 | cut -b$ccol)
  for i in $lst
  do
    if [[ $i = $nday ]]
```

```
    then
      echo $(cal $month $year | head -n1 )
    fi
  done
done
```

Первый аргумент программы – число, второй – номер дня недели.

Пример 2.

Написать *shell*-программу wcleaner для интерактивного удаления файлов. Программа должна позволять просматривать каталоги, устанавливать каталог для удаления файлов и показывать его путевое имя, удалять указанные файлы. Обеспечить возможность за один запуск удалить файлы из нескольких каталогов.

```
function help
{
echo "Usage: wfcl opton [files] [option] [files]"
echo "-l        list current directory"
echo "-p        print working directiry"
echo "-c {file} remove files"
echo "-g dir    go to dir"
echo "-h        this help"
}

cleaner ()
{
  j=0
  for i in $*
  do
    case $i in
    -*) break;;
    *) rm -i $i ;;
    esac
    j=$((j+1))
done
    return $j;
}

#Begin program

if [[ $# < 1 ]]
then
    help
fi

while  [[ $# > 0 ]]
do
```

```
  case $1 in
  "-h") help;;
  "-l") ls -la ;;
  "-p") pwd ;;
  "-g") shift
       if [[ $# > 0 ]]; then cd $1; else break; fi
        ;;
  "-d") shift
        cleaner $*
        shift $?
        continue
        echo $* ;;
    *)  echo default;;
  esac;
  shift;
done
```

Обратите внимание на разные способы задания функций, передачу параметров функции и возврат значения, обработку аргументов в циклах *for* и *while*, запись условного оператора в строку.
Программа интерпретирует опции слева направо, что позволяет указать группу команд:

```
wfcln -p -g dir1 -p -d file1 file2 -g ../dir2 \
-p -l -d "*.out" -l
```

Следует отметить, что несмотря на изменения текущего каталога в ходе выполнения программы, после окончания её работы текущий каталог остаётся прежним. Маска файлов должна передаваться в кавычках, иначе файлы будут выбраны из стартового каталога программы.

Упражнения

5.1. Присвойте переменной *myvar* некоторое текстовое значение. Напечатайте этот текст, используя переменную.

5.2. Присвойте переменной *MY_UNIX* название реализации операционной системы, в которой Вы работаете.

5.3. Присвойте переменной *IAM* Ваше имя регистрации. Запустите еще один *shell*. Видите ли Вы эту переменную? Что нужно сделать, чтобы увидеть ее в порожденном *shell*? Измените значение переменной *IAM* в порожденном *shell*. Выйдите из порожден-

ного *shell*. Посмотрите значение этой переменной в исходном *shell*. Объясните результат.

5.4. Создайте с помощью редактора *vi* файл *shell*-программы. Что нужно сделать, чтобы *shell* смог его выполнить?

5.5. Запустите *Bourne shell*. Как запустить на выполнение shell-программу в *Korn shell*? Укажите два способа.

5.6. Напишите *shell*-программу *privet*, которая будет:

выводить на экран некоторое приветствие с помощью команды *banner*;

присваивать Ваше регистрационное имя переменной *MYNAME*;

выводить значение ранее определенной переменной *MYNAME*;

выводить время и дату;

выводить имена всех пользователей, работающих в системе.

5.7. Перейдите в каталог */tmp*. Вызовите программу *privet*. Сможет ли *shell* найти программу? Как запустить программу на выполнение из */tmp*?

5.8. Напишите *shell*-программу *reverse*, которая будет принимать 12 аргументов и выводить их списком в обратном вводу порядке.

5.9. Напишите *shell*-программу *info*, которая будет просить пользователя ввести:

имя;

адрес;

число, месяц и год рождения;

и выводить эту информацию в следующем порядке:

имя;

год, число и месяц рождения;

адрес.

5.10. Задайте в качестве переменной *Y* некоторую строку. Используя команду *test*, определите, является ли *Y* строкой *UNIX*.

5.11. Выведите на экран список всех файлов указанного типа, имена которых начинаются на буквы *A*, *a* или имеют в названии слово *my*.

5.12. Напишите *shell*-программу, которая будет проверять, существует ли в текущем каталоге заданный файл.

5.13. Напишите *shell*-программу, которая принимает в качестве первого аргумента имя файла, а в качестве второго – имя каталога и проверяет, существует ли заданный файл в указанном каталоге. Если файл существует, то программа сообщает его тип. Если второй аргумент отсутствует, то файл следует искать в текущем каталоге.

5.14. Напишите *shell*-программу, включив в нее конструкцию *if*, которая будет выводить на экран *yes*, если переданный аргумент равен *UNIX*, и *no* в противном случае.

5.15. Напишите *shell*-программу, которая будет выводить на экран приглашение на ввод числа, сохранять введенное число в переменной *Y*, и печатать сообщение *Y is greater than 7,* если значение *Y* больше 7, и *Y is not greater than 7* в противном случае.

5.16. Напишите *shell*-программу, которая выводит на экран свои аргументы в отсортированном порядке.

5.17. Напишите *shell*-программу, которая выводит на экран следующую статистику:

а) свое имя;

б) количество аргументов, с которыми она запущена;

в) печатает каждый свой аргумент и длину аргумента в символах;

5.18. Напишите *shell*-программу, которая определяет количество аргументов в командной строке и выдает сообщение об ошибке, если количество аргументов не равно трем, или сами аргументы, если их количество равно трем.

5.19. Напишите *shell*-программу, которая бы запрашивала ввод с экрана трех имен в одной строке и выводила бы их в отсортированном порядке. При попытке ввести более трех имен программа должна определить, сколько имен введено, и напечатать избыточные имена в столбик с префиксом *extra>*.

5.20. Напишите shell-программу, которая выдает приветствие пользователю в зависимости от времени:

Good morning – с 6.00 до 12.00;
Good afternoon – с 12.00 до 18.00;
Good evening – с 18.00 до 22.00;
Good nigh – с 22.00 до 6.00.

5.21. Напишите программу, которая для файлов, указанных в командной строке, проверяет, существуют ли эти файлы, и определяет их тип (каталог, файл, канал, ссылка).

5.22. Напишите *shell*-программу, которая будет выдавать приглашение на ввод идентификатора пользователя, проверять идентификатор на соответствие используемым в системе и выводить полное имя домашнего каталога или, в случае недопустимого идентификатора, выдавать сообщение об ошибке.

5.23. Напишите *shell*-программу, которая будет запрашивать пользователя о том, хочет ли он увидеть содержимое текущего каталога, и выводить содержимое текущего каталога в случае положительного ответа. В случае отрицательного ответа запросите пользователя, содержимое какого каталога он хотел бы увидеть, и выведите содержимое требуемого каталога. Отработайте ситуацию, когда требуемый каталог не существует.

5.24. Напишите программу *mycopy*, которая будет копировать содержимое одного файла в другой. Аргументами командной строки будут файл-источник и файл-приемник. Предусмотрите следующие случаи:

файл-приемник не существует;

файл-приемник существует, и при записи его содержимое теряется;

файл-приемник существует, и при записи содержимое файла-источника добавляется в конец файла-приемника.

5.25. Задайте в качестве значения переменной *Y* некоторое число. Определите, на сколько *Y* больше 7. Выполните упражнение, не создавая файла сценария.

5.26. Напишите shell-программу *hello*, обеспечивающую следующую реакцию на аргументы командной строки:

аргумент *-d* – программа будет выполнять команду *date*;

аргумент *-w* – программа выведет список работающих пользователей;

аргумент *-l* – программа выведет содержимое текущего каталога;

при отсутствии аргументов или при неправильных аргументах в командной строке программа будет выводить справку о своих опциях.

5.27. Напишите *shell*-программу, удаляющую из указанного файла все повторяющиеся строки и сортирующую оставшиеся.

5.28. Напишите *shell*-программу, выдающую на экран отсортированный список дублирующихся в заданном файле строк.

5.29. Напишите программу, которая выводит на экран приглашение на ввод пяти чисел, находит и выводит их сумму.

5.30. Измените программу суммирования из задачи 5.29 таким образом, чтобы количество суммируемых чисел задавалось как аргумент командной строки.

5.31. Напишите программу-калькулятор, которая бы вводила либо число, либо знак операции, а в качестве приглашения выдавала текущее вычисленное значение.

5.32. Напишите программу *words*, которая будет выдавать пользователю приглашение на ввод по одному слову до тех пор, пока он не введет слово *end*. Запомните все введенные слова. После ввода слова *end* выведите на экран все введенные слова.

5.33. Измените программу *words* так, чтобы введенные слова выводились на экран в алфавитном порядке.

5.34. Создайте *shell*-программу для рассылки файла-почты всем пользователям, работающим в системе. Предусмотрите подтверждение для каждого пользователя.

5.35. Напишите программу, которая будет в течение минуты выводить сообщение на Ваш терминал каждые пять секунд.

5.36. Напишите *shell*-программу *newuser*, которая будет выполняться в фоновом режиме и каждые 20 секунд проверять, вошел ли в систему некий пользователь. Его имя должно передаваться в *newuser* через аргумент командной строки. Когда такой пользователь войдет в систему, программа должна выдать на экран Вашего терминала соответствующее сообщение.

5.37. Модифицируйте программу *newuser* так, чтобы имя контролируемого пользователя задавалось как аргумент командной строки, а программа записывала в файл протокола как время обнаружения входа пользователя, так и время его выхода.

5.38. Напишите *shell*-программу, которая будет перемещать все файлы, которые Вы хотите удалить, в подкаталог *.recycler* Вашего домашнего каталога. Перед перемещением файла не забудьте проверить, существует ли подкаталог *.recycler*, если нет – создайте его. Предусмотрите следующие опции:

-l – одновременный вывод на экран содержимого каталога *.recycler*;

-a – вывод суммарной памяти, занимаемый файлами каталога *.recycler,*

-d – очистка мусорной корзины (удаление всех файлов из каталога *.recycler*).

5.39. Измените приглашение системы так, чтобы оно содержало полное маршрутное имя текущего рабочего каталога.

5.40. Напишите программу, которая запрашивает имя пользователя и выводит его характеристики (группа, имя личного каталога, его содержимое).

5.41. Напишите программу, выводящую на экран строки файла, номера которых лежат в указанном диапазоне. Имя файла, номера начальной и конечной строк диапазона задаются как аргументы командной строки.

5.42. Модифицируйте программу из задачи 5.41 так, чтобы номера начальной и конечной строк вводились бы по запросу.

5.43. Замените в именах всех файлов в текущем каталоге начальную букву *a* на *A*.

5.44. Создайте каталог *text*. Переместите все файлы, имена которых имеют окончание *.txt,* в этот каталог, используя символы генерации имен файлов.

5.45. Измените расширение файлов *.txt* на расширение *.doc*.

5.46. Измените расширение файлов *.doc* на расширение *.text* только для тех файлов, имя которых состоит из пяти символов (включая расширение) и начинается на *a*.

5.47. Переименуйте все выполняемые файлы, добавив к их именам суффикс (расширение) *.exe*, но только в том случае, если имя файла еще не имеет такого суффикса.

5.48. Напишите программу, которая ищет самый большой файл в некотором указанном каталоге и выводит его полный маршрут.

5.49. Напишите программу, создающую подкаталоги в интерактивном режиме. Предусмотрите возможность перехода в другой каталог при вводе некоторого специального символа.

5.50. Напишите программу, устанавливающую Ваши *shell*-программы в указанный каталог.

5.51. Модифицируйте программу из задачи 5.50 так, чтобы она сохраняла указанное имя (описатель) программы в специальном файле и по некоторой опции выводила полный список установленных ею программ.

5.52. Модифицируйте программу из задачи 5.51 так, чтобы она удаляла установленную программу по указанному имени и одновременно удаляла ее из списка.

5.53. Напишите программу, определяющую наибольшее число из 10 аргументов командной строки.

5.54. Модифицируйте программу из задачи 5.53 так, чтобы количество чисел для сравнения передавалось как аргумент командной строки, а сами числа вводились пользователем в ответ на запрос.

5.55. Напишите программу, которая выдавала бы запрос на ввод чисел и после каждого ввода выдавала сообщение, лежит ли число в указанном диапазоне, до тех пор, пока в ответ на запрос пользователь не введет *end*. Верхнюю и нижнюю границы диапазона задайте как аргументы.

5.56. Модифицируйте программу из задачи 5.55 так, чтобы диапазоны задавались как аргументы, их количество вычислялось, а число проверялось на принадлежность каждому диапазону.

5.57. Напишите программу, определяющую наибольшее число среди записанных в файле. Имя файла передается как аргумент командной строки. Команду *sort* не использовать.

5.58. Присвойте переменной *line5* значение пятой строки файла */etc/passwd*.

5.59. Напишите *shell*-программу, которая определяет число символов в самом длинном своем аргументе и общее число символов во всех переданных ей аргументах.

5.60. Напишите *shell*-программу, которая вводит два слова и определяет, какое из них длиннее.

5.61. Напишите *shell*-программу, которая находит в указанном каталоге и его подкаталогах файл с самым длинным именем.

5.62. Модифицируйте решение задачи 5.61 так, чтобы поиск производился только среди файлов тех пользователей, имена которых указаны как аргументы программы.

5.63. Определите число символов, составляющих переменную *PATH*.

5.64. Определите, сколько каталогов включено в переменную *PATH*.

5.65. Выведите на терминал каждый каталог, указанный в переменной *PATH*, отдельной строкой.

5.66. Определите максимальную вложенность каталогов, указанных в *PATH*.

5.67. Напишите *shell*-программу, выводящую на терминал текущий каталог, полный путь до файла программы (исключая ее имя) и в отдельной строке – имя файла программы.

5.68. Напишите *shell*-программу, которая для заданного файла выводит его имя, размер, время последней модификации, время создания и время последнего доступа.

5.69. Для всех файлов из текущего каталога, содержащих в своем имени одно вхождение каждого из следующих трех символов: «точка», *a, b* (в произвольном порядке, в том числе и отделенные другими символами), выведите на экран таблицу следующего вида:

`Имя файла № позиции “.” № позиции “a” № позиции “b”.`

5.70. Напишите программу, которая формирует нулевой код возврата, если не получает аргументов, и ненулевой в противном случае. Как проверить правильность работы программы?

5.71. Напишите программу, которая подсчитывает количество доступных ей *shell*-переменных (имеющих значения), и количество переменных, не доступных порожденным процессам.

5.72. Напишите *shell*-программу, которая выводила бы число строк для тех файлов, в имени которых есть хотя бы один из символов, заданных в командной строке.

5.73. Напишите *shell*-программу – базу телефонных номеров. Программа позволяет добавлять пары «владелец–номер», по имени владельца выдавать номер, для заданного имени владельца сообщать, есть ли в базе номер, удалять или изменять запись.

5.74. Напишите программу *virus*, которая создает свою выполняемую копию с другим именем, а затем сама себя удаляет.

5.75. Напишите программу *virus2*, которая ищет в текущем каталоге программы командного языка и добавляет в их текст команду вывода на экран слова *Infected!*.

5.76. Напишите программу *virus3*, которая добавляет к найденным ею программам командного языка свой код для заражения других программ.

5.77. Напишите программу *virus4*, модифицировав результат задачи 5.76 так, чтобы при заражении заражающий код удалялся из заражающей программы.

5.78. Напишите программу *antivirus*, которая бы находила все зараженные программой из задачи 5.77 *shell*-программы.

5.79. Модифицируйте разработанную в задаче 5.78 программу так, чтобы она не только находила зараженные программы в указанном ей каталоге, но и «вылечивала» бы их, сохраняя заражен-

ную версию в новом файле, к имени которого добавлено окончание *.vir*, и снимая с такого файла атрибут выполнения.

5.80. Напишите программу *virus5*, которая бы:

а) заражала бы вирусом из задачи 5.78 *shell*-программы в текущем каталоге и его подкаталогах, только если пользователь вводил команду *ls*;

б) вела бы себя как команда *ls*, ничем не выдавая на терминале своей работы.

5.81. Напишите программу *virus6* на основе программ *virus2-virus5*, заражающий код которой нельзя было бы обнаружить разработанной Вами программой *antivirus* из задачи 5.79, а заражение происходило бы при любом значении переменной *PATH*.

5.82. Напишите программу *supervirus*, запуск которой заражает Ваши файлы вирусом, разработанным в предыдущей задаче, а при каждом Вашем входе в систему делается попытка заразить файлы Ваших товарищей. Программа *supervirus* стирает себя после первого запуска.

5.83. На основе результатов задачи 5.79 напишите программу *superantivirus*, которая обнаруживает и полностью «излечивает» (если указана соответствующая опция) все файлы в указанном каталоге и его подкаталогах от всех вирусов, разработанных в задачах 5.75 – 5.82.

6. УПРАВЛЕНИЕ ПРОЦЕССАМИ. СИГНАЛЫ

Интерпретатор команд обеспечивает развитые инструменты управления процессами, предоставляя пользователю следующие средства.

Встроенные специальные переменные:

? (вопросительный знак) содержит код завершения последней выполненной команды;

$ (знак доллара) содержит *PID* (числовой идентификатор) процесса, выполняющего *shell*;

! (восклицательный знак) позволяет узнать *PID* последней асинхронной команды (фонового процесса, запущенного последним);

PPID инициализируется идентификатором родительского процесса.

Средством синхронизации процессов служит встроенная команда *wait*. Вызванная без аргументов, она ожидает завершения всех дочерних процессов. В качестве аргумента может быть указан идентификатор ожидаемого процесса или идентификатор задания. Команда возвращает код завершения ожидаемого процесса. Кодом завершения *shell*-программы является код последней выполненной команды, но его можно сформировать принудительно, указав желаемый код аргументом *exit*. При обычном завершении эта величина не превышает 128. Если процесс прерывается сигналом, его код содержится в возвращаемом значении. Определить сигнал можно, например, так:

```
myprocess&
pid=$!
kill -KILL $pid
wait $pid
echo Terminated by a SIG$(kill -l $(($?-128)))
```

Сигналы. В управлении процессами сигналы играют важнейшую роль, причем наряду с операционной системой их полезно применять и пользователю (префикс *SIG* опущен).

Сигналом *KILL* процесс уничтожается немедленно. Ему не дается времени на «аккуратный» останов. Процесс даже не извещается о сигнале, скорее это способ указания операционной системе. Другой сигнал, который так же, как и *KILL,* не может быть проигнорирован или перехвачен, – это *STOP*, останов процесса. Продолжить его исполнение можно посылкой сигнала *CONT*. Обычно для остановки процесса *shell* по команде «интерактивного

останова» <*Ctrl+Z*> использует сигнал *TSTP*. Сигнал *CHLD* указывает на изменение статуса дочернего процесса.

Для «вежливого» завершения процесса, т. е. с уведомлением о необходимости закончить работу, используется сигнал *TERM*. Интерактивное прерывание командой <*Ctrl+C*> приводит к посылке сигнала *INT* – «прерывание процесса». Сигнал *QUIT* аналогичен, его можно сгенерировать нажатием <*Ctrl+|*>, но такое завершение приводит по умолчанию к созданию файла *core*.

Обычно при принудительном завершении процесса в сценариях, использующих временные файлы, существует необходимость их удаления. Такой код может быть описан в ловушке *trap*, связанной с перечисленными выше сигналами. Недостатком такого подхода является возможность как забыть какой-либо сигнал, так и продублировать этот же код при нормальном завершении сценария. В таком случае удобно использовать ловушку для сигнала *EXIT* (нуль-сигнал, 0). Эта ловушка по сути выполняет роль обработчика завершения, исполняя код в любом случае – как при нормальном завершении, так и при завершении по сигналу.

Для собственных нужд пользователь может использовать *USR1* и *USR2*, самостоятельно наделяя их смыслом.

Управление заданиями. Процессами можно управлять не только посылкой им сигналов, но и интерактивно с использованием механизма управления заданиями: команды *jobs, bg, fg*. Однако и сам процесс может остановиться, используя команду *suspend*, правда возобновить работу без помощи другого процесса он не сможет. Команда *stop* останавливает указанный процесс. Идентификаторы системы управления заданиями:

%число – номер задания;
%строка – начальная часть командной строки;
%?подстрока – задание, содержащее в команде подстроку;
%% или %+ – текущее задание;
%- – предыдущее задание.

Указанные обозначения могут быть использованы в качестве аргументов *wait*.

Приоритеты процессов. Наряду с предоставлением средств управления собственно исполнением процессов, имеются гибкие средства управления «скоростью» работы. Наряду с широко известными командами *nice, renice, nohup*, непосредственно из командной строки доступны изменения не только приоритета, но и

класса процесса (например, перевод его из процесса с разделением времени в процесс реального времени), кванта времени в многозадачной среде (команда *priocntl*).

В раздел включены упражнения на следующие темы:

- наблюдение за процессами (*ps*);
- запуск программ в фоновом режиме (*&, nohup*);
- перевод задания из фонового режима в оперативный и наоборот (*bg, fg*);
- запуск программ с разными приоритетами, изменение приоритетов программ (*nice, renice*);
- ожидание завершения процесса (*wait*);
- задание среды выполнения процессов (*env, export, set*);
- управление процессами с помощью сигналов (*kill, trap*).

Пример 1. Программа, для остановки которой требуется нажать ^C три раза.

```
trap react INT
react()
{
  n=$(($n+1))
  if [[$n < 2 ]]
  then
    exit 1
  fi
}
n=0
while true
do
  echo -e ". \c"
  sleep 1
done
```

Пример 2. Написать программу, запускающую переданную ей как аргумент программу, которая не позволяет ей работать более 5 секунд.

```
trap liller USR1
killer()
{
echo Time exceeded
kill -9 pid
```

```
}
./ttimer 5 &
pid2-$!
$1 &
pid=$!
wait $pid
v=$(ps | grep $pid2)
if [[ $v != “” ]]
then
  kill -9 $pid2
fi
---------Вспомогательная программа ttimer-------
sleep $1
v=$(ps | grep $pid2)
if [[ $v != “” ]]
then
  kill -s USR1 $PPID >/dev/null
fi
```

Упражнения

6.1. Напишите *shell*-программу, выполняющую бесконечный цикл, одну итерацию в секунду. Запустите ее в фоновом режиме. Остановите программу.

6.2. Напишите программу, которая выдает в бесконечном цикле сообщение, заданное как аргумент. Как остановить такую программу?

6.3. Запустите несколько программ из одного терминального окна. Как переключиться на заданную программу?

6.4. Запустите программу из задачи 6.1 с пониженным приоритетом. Как повысить ее приоритет? Как запустить несколько таких программ одной командной строкой? Как переключаться между ними? Остановите запущенные Вами программы.

6.5. Запустите вновь программу из задачи 6.1 в фоновом режиме и отключитесь от системы. Вновь зарегистрируйтесь. Работает ли Ваша программа? Какую команду нужно использовать, чтобы запущенная Вами в фоновом режиме программа продолжала работать после Вашего выхода из системы?

6.6. Напишите *shell*-программу, которая не прекращает своего выполнения при выходе из системы.

6.7. Модифицируйте программу из задачи 6.1 так, чтобы при ее запуске создавался файл *infinite.tmp*, который удаляется при завершении программы со стороны пользователя.

6.8. Напишите программу, которая сообщает имя переданного ей сигнала и записывает его в файл с указанием времени события.

6.9. Напишите программу *manager*, которая запускает две другие написанные Вами программы с разными приоритетами. Эти программы должны периодически добавлять в файлы некоторую информацию разного объема. При превышении заданного объема одним из файлов *manager* прекращает выполнение обеих программ и выводит имя той из них, которая создала больший файл.

6.10. Модифицируйте предыдущую программу так, чтобы приоритеты и ограничение на размер файлов вводились как аргументы командной строки.

6.11. Модифицируйте предыдущую программу так, чтобы она выводила размер созданных файлов и по некоторому сигналу удаляла бы их.

6.12. Напишите программу-диспетчер, которая запускала бы две другие *shell*-программы, каждая из которых выводила бы в цикле сообщение о своей работе. Цель программы-диспетчера – обеспечить псевдопараллельную работу этих программ, поочередно выделяя квант времени для каждой, но так, чтобы в каждый момент времени работала бы только одна программа.

6.13. Модифицируйте программу-диспетчер из предыдущей задачи так, чтобы приоритет программ (длительность квантов времени) можно было задавать в командной строке для каждой программы.

6.14. Установите в качестве первичного приглашения идущие часы, меняющие свои показания каждые 10 секунд.

6.15. Напишите программу, которая приближенно подсчитывает время своей работы, увеличивая в цикле счетчик секунд, а при нажатии пользователем <*Ctrl*+*C*> сообщает время (в секундах), в течение которого она проработала.

6.16. Напишите программу-будильник, которая работает в фоновом режиме и в указанное время напоминает пользователю о важном деле.

6.17. Напишите программу, которая, работая в фоновом режиме, в указанное время отключает пользователя от системы.

6.18. Напишите *shell*-программу, которая в фоновом режиме запускает другую *shell*-программу, указанную ей в качестве аргумента, печатает сообщение:

`«имя запущенной программы» already started`

и ждет завершения программы, после чего сообщает:

`«имя запущенной программы» finished`.

6.19. Напишите программу, которая запускает в фоновом режиме все процессы, указанные как аргументы, и ждет завершения последнего из них, после чего сообщает, что все процессы завершены. В качестве дочерних программ используйте написанные Вами *shell*-программы с различной длительностью выполнения. Предусмотрите возможность передачи программам разного количества аргументов.

6.20. Модифицируйте решение задачи 6.19, чтобы родительская программа дожидалась окончания только первой и второй из запущенных ею программ.

6.21. Напишите *shell*-программу, которая запускает дочернюю *shell*-программу и ждет ее завершения. Если через указанное время дочерняя программа не завершилась, родительская программа прекращает ее выполнение.

6.22. Напишите *shell*-программу, запускающую дочернюю shell-программу и завершающую свое выполнение через заданное вре-

мя, но до завершения дочерней программы. Каковы значения идентификатора родительского процесса дочерней программы до и после завершения родительской программы?

6.23. Модифицируйте решение задачи 6.22 так, чтобы спустя заданное время дочерний процесс завершал родительский, если он к этому времени не завершался сам.

6.24. Напишите группу *shell*-программ, взаимодействующих следующим образом. Основная программа запускает в фоновом режиме две *shell*-программы. Первая дочерняя программа подсчитывает число файлов в текущем каталоге, а затем переходит в состояние ожидания, пока вторая дочерняя программа не завершит подсчет слов в указанном файле, после чего дочерние программы завершаются вместе.

6.25. Напишите группу *shell*-программ, взаимодействующих следующим образом. Основная программа запускает в фоновом режиме две дочерние программы. Первая дочерняя программа подсчитывает строки в некотором указанном файле, а вторая – подсчитывает слова в другом файле. Первая завершившая работу программа ожидает другую, а затем программы «меняются» файлами и проделывают ту же работу, начав ее одновременно.

6.26. Напишите группу *shell*-программ, взаимодействующих следующим образом. Основная программа запускает в фоновом режиме пять дочерних программ, каждая их которых выполняет свою работу, состоящую из двух этапов. Второй этап все дочерние программы начинают одновременно. Если хотя бы одна дочерняя программа не успела завершить свою работу за заданное время, родительская программа уничтожает все дочерние процессы.

7. ПРАВА ДОСТУПА И ЗАЩИТА ФАЙЛОВ

Традиционная модель защиты файлов *UNIX*, построенная на установлении прав доступа для владельца, группы и остальных пользователей, довольно проста, но не достаточно гибка и удобна для применения в больших многопользовательских системах. В ряде систем реализована возможность защиты файлов с использованием списков управления доступом (*ACL – access control lists*). Становится возможным индивидуально устанавливать права доступа. К сожалению, реализация этой возможности в разных системах различна.

В системе *HP-UX* используются команды *lsacl* и *chacl*. Доступ к файлу указывается комбинациями *(пользователь.группа, режим)*. Символ *%* означает любого пользователя или группу. Для каждого файла можно задать до 13 пар прав доступа. Посмотреть установленные наборы можно с использованием *lsacl*, а изменить или удалить с помощью *chacl*.

В ОС *Solaris* и *Linux* для этих целей используются команды *getfacl* и *setfacl* (что соответствует стандарту POSIX).

С целью обратной совместимости с программами, не поддерживающими ACL, реализована возможность задания маски, позволяющей ограничить списки доступа пользователей и групп, а также установить списки прав по умолчанию на каталог (следует предварить элемент списка префиксом *d:*), что позволит создаваемым в нем файлам и каталогам получить права доступа по умолчанию (включая маску). В этом случае *umask* не используется. Пример вывода *getfacl* для каталога:

```
# file: dir1
# owner: u1
# group: staff
user::rwx
group::rwx   #effective:rwx
mask:rwx
other:---
default:user::rw-
default:user:u3:rwx
default:group::r--
default:group:g1:---
default:mask:rwx
default:other:---
```

Символ # считается началом комментария до конца строки. Эффективные права являются результатом ограничения по маске, что даёт удобный инструмент временного ограничения прав. Кроме того ключевые слова, такие как user, могут быть заменены своей первой буквой.

Определить файлы и каталоги, для которых установлены списки управления доступом, можно по выводу команды ls –l. Список прав доступа для таких файлов оканчивается знаком +.

Пример 1. Для файла *file* установить владельцу полный доступ, а другим пользователям запретить выполнение программы *file*:

```
setfacl -s u::rwx,g::---,o::---,m:rw- file
```

Пример 2. Для файла *file* установить дополнительно право на чтение пользователю *u1*, пользователю *u2* и членам группы *g1* – право на чтение и запись:

```
setfacl –m u:u1:r-- ,u:u2:rw-,g:g1:rw- file
```

Пример 3. Установить права доступа к файлу file2 такими же, как и к файлу file1:

```
getfacl file1 | setfacl -f - file2.
```

Пример 4. Пользователю *u1* предоставить возможность читать все Ваши файлы. Данный пользователь входит в группу *g1*, которой не принадлежит ни один файл пользователя *u1*. Файловая система не поддерживает *ACL*.

```
cp /usr/bin/cat mycat
chown u1 mycat
chgrp g1 mycat
chmod 4710 mycat
```

Пользователь *u1* должен будет использовать *mycat* для доступа к файлам. Команда chown приведена для наглядности.

При выполнении заданий данного раздела потребуются следующие команды:

- *chmod* – изменение прав доступа к файлам;
- *chown* – изменение владельца файла;

- *chgrp* – изменение группы;
- *umask* – маскирование прав доступа;
- *setfacl* – установка режима доступа к файлу с помощью списков управления доступом (ОС Solaris);
- *getfacl* – просмотр режимов доступа, установленных с помощью списков управления доступом (ОС Solaris);
- *su* – переключение идентификатора пользователя;
- *newgrp* – переключение идентификатора группы.

Упражнения

7.1. Создайте файл с помощью команды *cat.* Кто и какие права доступа имеет к этому файлу? Можете ли Вы прочитать этот файл?

7.2. Установите права доступа к файлу -*w*- --- ---. Сможете ли Вы теперь прочитать этот файл?

7.3. Установите права доступа к файлу *rw*- --- ---. Можете ли Вы теперь прочитать свой файл? А Ваш товарищ? Как нужно изменить права доступа, чтобы Ваш товарищ мог прочитать файл?

7.4. Задайте маску, разрешающую запись и изменение файла только Вам. Создайте новый файл. Попросите товарища удалить Ваш файл. Какова реакция системы на его попытку? Можете ли Вы удалить свой файл?

7.5. Что нужно сделать, чтобы созданные Вами с помощью vi файлы shell-программ были не доступны для выполнения и чтения другим пользователям?

7.6. Измените права доступа к своему домашнему каталогу так, чтобы Ваш товарищ не смог прочитать его содержимое или записать что-либо.

7.7. Измените права доступа к своему домашнему каталогу так, чтобы Ваш товарищ мог читать его содержимое, но не мог изменять его.

7.8. Создайте файл с правами доступа *rw- r-- ---*. Назначьте владельцем файла Вашего товарища. Есть ли у Вас доступ к файлу? Можете ли Вы снова стать владельцем файла?

7.9. Кто является владельцем файла */etc/passwd*? К какой группе он принадлежит? Кто имеет право изменить владельца или группу? Можете ли Вы внести изменения в файл?

7.10. Создайте копию файла */etc/passwd*. Можете ли Вы внести изменения в файл-копию?

7.11. Определите права доступа к файлу устройства, ассоциирующегося с Вашим терминалом. Кто является владельцем этого файла? Выполните команды *mesg y* и *mesg n*, одновременно следя за изменениями прав доступа к этому файлу. Что фактически делает команда *mesg*?

7.12. Обеспечьте возможность вывода на Ваш терминал сообщений только некоторого определенного пользователя.

7.13. Находясь в домашнем каталоге, создайте каталог *T* с непустым подкаталогом *F*. Установите права доступа к каталогу *T r-- --- ---*. Посмотрите содержимое каталога *T*. Посмотрите содержимое каталога *F*. Объясните результат.

7.14. Установите права доступа к каталогу *T --x --- ---*. Посмотрите содержимое каталога *T*. Посмотрите содержимое каталога *F*. Объясните результат.

7.15. Напишите программу, которая для файлов, указанных в командной строке, устанавливала бы бит выполнения для владельца.

7.16. Создайте каталог, содержимое которого Вы можете распечатать, но не можете ни удалить файл, ни создать новый.

7.17. Создайте каталог, в котором Вы можете создавать новые файлы или удалять существующие, но содержимое которого Вы не можете посмотреть.

7.18. Создайте каталог, в который Ваши товарищи могут копировать свои файлы, но содержимое которого они не могут посмотреть. Вы же можете посмотреть содержимое этого каталога.

7.19. Можно ли разрешить копировать файл из Вашего каталога только трем Вашим товарищам? Если да, то как это сделать?

7.20. Обеспечьте Вашему товарищу возможность создать в Вашем каталоге файл нулевого размера, запись в который Вашему товарищу запрещена и владельцем которого являетесь Вы.

7.21. В каталоге из задачи 7.18 создайте новый каталог. Обладает ли он свойствами родительского? Почему? Что нужно сделать, чтобы эти свойства были у каталога по умолчанию после его создания?

8. КОМАНДА find. РЕЗЕРВНОЕ КОПИРОВАНИЕ

Для выполнения заданий данного раздела потребуются команды:

- *find* – поиск файлов, удовлетворяющих заданным требованиям;
- *cpio* – копирование данных со стандартного ввода на стандартный вывод;
- *tar* –- создание архива на магнитной ленте и восстановление из архива;
- *dd* –- бинарное копирование файлов.

Пример 1. Найти и стереть текстовые файлы и их резервные копии, последнее изменение которых было более недели назад

```
find ~ \( -name *.bak -o -name '*.txt' \) \
     -mtime +7 -exec rm {} \;
```

Пример 2. Создать архив домашнего каталога

```
cd ~
tar -cvf homearchive.tar * .*
```

Извлечь из архива

```
tar -xvf homearchive.tar
```

Упражнения

8.1. Создайте псевдоним *rmzero*, удаляющий из Вашего личного каталога все файлы нулевого размера.

8.2. Создайте псевдоним *rmold*, предлагающий к удалению все файлы с расширением *tmp*, к которым не было доступа по чтению в течение последнего месяца (с подтверждением каждого удаления).

8.3. Создайте архив всех файлов с расширением *doc* в Вашем личном каталоге.

8.4. Используя команды *find* и *cpio*, создайте копию какого-либо подкаталога Вашего личного каталога. Удалите несколько файлов из Вашего личного каталога. Восстановите удаленные файлы из архива-копии.

8.5. Создайте псевдоним *arcold*, создающий архив всех файлов, к которым не было доступа по чтению в течение последнего месяца.

8.6. Выдайте на экран список всех файлов из заданного каталога и его подкаталогов, к которым не было доступа более восьми дней.

8.7. Выдайте на экран список всех файлов из заданного каталога и его подкаталогов, которые изменились за последние восемь дней.

8.8. Уничтожьте все файлы в своих каталогах и подкаталогах, созданные более 30 дней назад (будьте осторожны, не удалите *.profile* и другие файлы настройки).

8.9. Найдите все файлы с числом ссылок, равным двум.

8.10. Создайте архив всех файлов заданного пользователя, к которым не было доступа более девяти дней.

8.11. С использованием команды *tar* передайте другому пользователю копию дерева подкаталогов своего каталога вместе с файлами. Попросите другого пользователя передать Вам копию дерева своего каталога. Восстановите каталог вашего товарища в своем каталоге. Кто является владельцем файлов?

8.12. Создайте архив одного из каталогов Вашего домашнего каталога. Разделите его на две части. Соедините их и восстановите данные из архива.

УКАЗАНИЯ К РЕШЕНИЮ ЗАДАЧ

1.2. *UNIX* различает регистры клавиатуры. Как правило, команды набираются в нижнем регистре.

1.12. Наименование соседнего терминала можно определить по имени регистрации пользователя, который за ним работает.

1.14. Используйте две команды *cal* с соответствующими аргументами в одной строке, объединенные операцией ;.

1.15.
```
echo -e \
"-----------------------------------------------\
\n\n\n"; cal 2007; echo -e "\n\n\n\
-----------------------------------------------"
```

Обратная косая черта использована, чтобы расположить командную строку на нескольких строчках ввода и облегчить подсчет символов разреза, разрывая её в нужном месте.

1.17. Можно ли послать сообщение одновременно на несколько терминалов, не используя цикл? См. указание к задаче 1.14.

1.18. Следует использовать *wall*.

1.24. Воспользуйтесь командой *ps*.

2.3. Каталог */export/home* характерен для ОС *Solaris*. Для других ОС семейства *UNIX – /home*.

2.14. Для создания файла, содержащего заданный текст, используйте команду *cat*, перенаправив ее вывод в файл:

```
cat > myfile <Enter>
ввод текста
<Ctrl+D>
```

2.20. Для добавления строки к файлу используйте команду *cat,* перенаправив ее выход в файл:

```
 mylink: cat >> mylink <Enter>
ввод текста
<Ctrl+D>
```

Можно использовать перенаправление команды *echo*:

```
echo “ текст” >> mylink
```

2.24. Чтобы псевдоним создавался всякий раз при открытии нового сеанса, следует поместить соответствующую команду *alias* в файл *.profile* Вашего личного каталога.

2.26. Помните, что файл с именем «точка» обозначает текущий каталог, а файл с именем «две точки» - родительский.

2.28. `alias joke="echo ALL FILES...;ls"`

2.29. `alias joke="echo ALL FILES...;ls;unalias joke"`

2.34. Удаление файла, имя которого состоит из четырех пробелов:
`rm -i \ \ \ \ .`

2.39 – 2.40. Символическая ссылка – специальный файл, содержащий путевое имя другого файла, поэтому замена файла на другой не оказывает влияния на саму ссылку. Напротив, жесткая ссылка связана с блоками данных файла и является одним из равноправных имён файла.

4.3. Командами `cat >>myfile` и `echo -e "...\n..." >>myfile`.

4.7.
```
cat >prompt;  alias mktxt="cat prompt;cat >"
  Please, type text
  ^D
```

4.11. Числовой идентификатор пользователя – восьмое поле в выводе команды *who -T.*

4.13. Для сохранения списка в файле используйте команду *tee*.

4.20. Предусмотрите ситуацию, когда один пользователь работает с нескольких терминалов.

4.23. Информация обо всех зарегистрированных в системе пользователях содержится в файле */etc/passwd*. У пользователей, которым запрещена регистрация, в поле «пароль» стоит символ *.

4.25. Для выделения строк с пятой по десятую используйте в конвейере команды *tail* и *head*.

4.27. `cat file1 - file2  >file3`

5.5. Первый способ – указать имя *shell*-программы как аргумент при вызове *Korn shell*:

```
ksh -c имя_программы
```

Второй способ – в первой строке *shell*-программы указать маршрутное имя до *Korn shell*:

```
#!/bin/ksh
```

5.7. Модифицируйте значение переменной среды *PATH* или укажите полное или относительное маршрутное имя.

5.9. Используйте *read*.

5.12, 5.13. Используйте *test*.

5.19. Используйте команду *read* с четырьмя аргументами, тогда четвертый аргумент будет содержать «лишние» аргументы, если они есть. Для распечатки «лишних» аргументов в столбик используйте цикл *for* с указанием значения четвертого аргумента в качестве диапазона значений переменной цикла.

5.25. Содержимое файла следует ввести в командной строке с использованием символов ; и \. Можно использовать функцию.

5.27, 5.28. Для обработки групп повторяющихся строк можно использовать программу *uniq*. Другой способ: в цикле сравнивать попарно строки из файла, читая файл построчно и запоминая каждую строку в переменной.

5.31. Используйте *case* для анализа ввода команды *read*.

5.35, 5.36. Используйте команду *sleep*.

5.39. Вид приглашения определяется переменной *PS1*, текущий рабочий каталог – переменной *PWD*.

5.41. Используйте в конвейере команды *head* и *tail*.

5.43. Используйте конструкцию *${i#шаблон}*.

5.44, 5.45. Используйте конструкцию *${i%шаблон}*.

5.46. Используйте конструкции *${i%шаблон}* и *${#переменная}*.

5.75. Для выделения файлов, содержащих *shell*-программы, используйте команду *file*. Не следует полагаться только на проверку, выполняемый ли файл, так как такой файл может содержать двоичный исполняемый код (программу).

5.81. Заражающий код следует поместить в функцию *ls*, размещенную в *.profile.*

6.8. Мнемонические имена сигналов и их расшифровка хранятся в файле */usr/include/sys/signal.h.*

6.12. Программа-менеджер поочередно посылает дочерним программам сигналы *STOP* и *CONT.*

6.13. Кванты времени определяются аргументом *sleep* в программе-менеджере.

6.14. В фоновом режиме следует запустить программу, посылающую сигнал, например, *USR1*, основной программе, которая сама формирует приглашение с использованием *date*, читает пользовательский ввод и передает его на выполнение порожденному *shell*.

7.12, 7.19. Используйте списки управления доступом.

7.20. Используйте копию программы *touch* с правами *--s--x---,* владельцем которой являетесь Вы.

8.1, 8.2. Используйте возможность указания команды в *find*.

8.11. Наряду с *tar* следует использовать *chown*.

8.12. Наряду с *tar* следует использовать *dd*.

СПИСОК ЛИТЕРАТУРЫ

1. Армстронг Д. Секреты UNIX. М.: Диалектика. 2000.

2. Банахан М., Раттер Э. Введение в операционную систему UNIX. М.: Радио и связь. 1986.

3. Баурн С. Операционная система UNIX. М.: Мир. 1986.

4. Браун П. Введение в операционную систему UNIX (МОЭ). М.: Мир. 1987.

5. Готье Р. Руководство по операционной системе UNIX. М.: Финансы и статистика. 1985.

6. Дайсон П. Операционная система UNIX: настольный справочник. М.: ЛОРИ. 1997.

7. Дегтярев Е.К. Введение в UNIX. М.: Память. 1992.

8. Забродин Л.Д. UNIX: введение в командный интерфейс. М.: Диалог-МИФИ. 1994.

9. Келли-Бутл С. Введение в UNIX. М.: ЛОРИ. 1995.

10. Керниган Б.В., Пайк Р. UNIX – универсальная среда программирования. М.: Финансы и статистика, 1992.

11. Кристиан К. Введение в операционную систему UNIX. М.: Финансы и статистика. 1985.

12. Пин Д., О'Райли Т., Лукидис М. UNIX: инструментальные средства. Киев: Изд. гр. BHV. 1998.

13. МакМален Д. UNIX. М.: Компьютер. ЮНИТИ. 1996.

14. Топхем Д., Хай Ван Чыонг. Юникс и Ксеникс. М.: Мир. 1988.

Сергей Викторович Ктитров

Наталья Владимировна Овсянникова

КОМАНДНЫЙ ЯЗЫК ОС UNIX

Лабораторный практикум

Редактор Т.В. Волвенкова

Подписано в печать 24.10.2007 Формат 60×84 1/16
Уч.-изд. л. 3,75 Печ. л.3,75 Тираж 150 экз.
Изд. № 3/3 Заказ №

Московский инженерно-физический институт
(государственный университет).
115409, Москва, Каширское шоссе, 31

Типография издательства «Тровант».
г. Троицк Московской обл.