Министерство образования Российской Федерации

Владимирский государственный университет

КОМПЛЕКСНАЯ ЗАЩИТА
ОБЪЕКТОВ ИНФОРМАТИЗАЦИИ

Ю.А. ИЛЛАРИОНОВ Д.А. ПОЛЯНСКИЙ

ВВЕДЕНИЕ В СПЕЦИАЛЬНОСТЬ

Практикум

Под редакцией кандидата технических наук
М.Ю. Монахова

Владимир 2003

УДК 519.6 (075)
И44

Рецензенты:

Кандидат технических наук, доцент
зав. кафедрой информатики и вычислительной техники
Владимирского государственного педагогического университета

Ю.А. Медведев

Доктор физико-математических наук, профессор
Владимирского государственного педагогического университета

Ю.А. Алхутов

Печатается по решению редакционно-издательского совета
Владимирского государственного университета

Илларионов Ю.А., Полянский Д.А.
И44 Введение в специальность: Практикум / Под ред. М.Ю. Монахова; Владим. гос. ун-т. Владимир, 2003. 76 с. (Комплексная защита объектов информатизации).

ISBN 5-89368-417-6

Это первая книга из серии «Комплексная защита объектов информатизации». В ней представлен систематизированный материал по основам построения и администрирования операционной системы UNIX.

Практикум предназначен для студентов специальности 075400 «Комплексная защита объектов информатизации» дневной формы обучения. Может быть полезен широкому кругу читателей, самостоятельно осваивающих вопросы информатики и вычислительной техники.

Ил. 11. Библиогр.: 11 назв.

УДК 519.6 (075)

ISBN 5-89368-417-6

ПРЕДИСЛОВИЕ

Пособие является первой книгой из серии «Комплексная защита объектов информатизации», подготовленной кафедрой «Информатика и защита информации» Владимирского государственного университета.

Учебный практикум предназначен в первую очередь для студентов специальности «Комплексная защита объектов информатизации», которые знакомство со своей будущей профессией начинают с «наиболее продвинутой» в плане безопасности операционной системы UNIX. В настоящее время существует ряд книг по UNIX. К сожалению, все они либо для профессионалов, либо для «некомпьютерщиков». Таких пособий, которые комплексно рассматривают проблемы защиты информации на уровне операционной системы, предназначенных для начинающих осваивать серьезную специальность, явно недостаточно.

Практикум познакомит с особенностями архитектуры операционной системы (ОС) UNIX на примере реализации ОС Linux, научит пользоваться основными командами языка управления системой (Shell), приемами редактирования текстов с помощью редактора KWord, а также даст вам необходимые базовые знания по администрированию и защите информации и поможет научиться приемам написания достаточно сложных программ с использованием интерпретатора командной строки.

Практикум состоит из четырех практических работ, каждая из которых выполняется под руководством преподавателя или самостоятельно (дистанционная форма обучения) и включает в себя теоретическое введение; упражнения, которые необходимо выполнить в диалоге с системой; практическое задание. Для самоконтроля приводятся вопросы.

ВВЕДЕНИЕ

UNIX - одна из самых популярных в мире операционных систем благодаря тому, что ее сопровождает и распространяет большое число компаний. Первоначально она была создана как многозадачная система для миникомпьютеров и мэйнфреймов в середине 70-х годов, но с тех пор она выросла в одну из наиболее распространенных операционных систем, несмотря на свой временами недружелюбный интерфейс и отсутствие централизованной стандартизации.

В чем реальная причина популярности UNIX? Дело в том, что из всех серверных систем, существующих в настоящее время, UNIX является самой стабильной и отлаженной. Отсюда и появление Linux как системы, разрабатываемой все более расширяющейся группой энтузиастов UNIX, которые хотят внести свою лепту в процесс ее создания.

Существуют версии UNIX для многих систем, начиная от персонального компьютера до суперкомпьютеров. Большинство версий UNIX для персональных компьютеров достаточно дороги и сложны. Linux - свободно распространяемая версия UNIX, первоначально была разработана Линусом Торвальдсом в университете Хельсинки (Финляндия). Linux был создан с помощью многих UNIX-программистов и энтузиастов из Internet.

Сегодня Linux - это полноценная ОС семейства UNIX, способная работать с X Windows, TCP/IP, Emacs, UUCP, mail и USENET. Практически все важнейшие программные пакеты были поставлены и на Linux, т.е. для Linux теперь доступны и коммерческие пакеты. Все большее разнообразие оборудования поддерживается по сравнению с первоначальным ядром. Таким образом, этот "маленький UNIX" вырос настолько, что может делать практически все в мире компьютеров.

Практическая работа № 1

ИССЛЕДОВАНИЕ СИСТЕМЫ КОМАНД UNIX

1. Цель работы

Получение практических навыков в использовании базовых команд операционной системы UNIX.

2. Теоретические сведения

2.1. Системные характеристики

Linux поддерживает большинство свойств, присущих другим реализациям UNIX, плюс ряд тех, которых больше нигде нет. Linux - это полная многозадачная многопользовательская операционная система (точно так же, как и другие версии UNIX). Это означает, что одновременно много пользователей могут работать на одной машине, одновременно выполнять много программ.

Linux достаточно хорошо совместим с рядом стандартов для UNIX (насколько можно говорить о стандартизации UNIX) на уровне исходных текстов. Большинство свободно распространяемых по сети Internet программ для UNIX может быть откомпилировано для Linux практически без особых изменений. Кроме того, все исходные тексты для Linux, включая ядро, драйверы устройств, библиотеки, пользовательские программы и инструментальные средства, распространяются свободно.

Другие специфические внутренние черты Linux включают контроль работ по стандарту POSIX (используемый оболочками, такими как csh и bash), псевдотерминалы (pty), поддержку национальных и стандартных клавиатур динамически загружаемыми драйверами клавиатур.

Linux также поддерживает виртуальные консоли (virtual consoles), которые позволяют "переключать экраны" на консоли в текстовом режиме.

Ядро может само эмулировать команды 387-FPU, так что системы без сопроцессора могут выполнять программы, на него рассчитанные (т.е. с плавающей точкой).

Linux поддерживает различные типы файловых систем для хранения данных. Некоторые файловые системы, такие как файловая система *ext2fs*, были созданы специально для Linux. Поддерживаются также другие типы файловых систем, такие как Minix-1 и Xenix. Реализована также файловая

система MS-DOS, позволяющая прямо обращаться к файлам MS-DOS на жестком диске. Поддерживается также файловая система ISO 9660 CD-ROM для работы с дисками CD-ROM.

Linux обеспечивает полный набор протоколов TCP/IP для сетевой работы. Это включает драйверы устройств для многих популярных карт Ethernet, SLIP (Serial Line Internet Protocol, обеспечивающие вам доступ по TCP/IP при последовательном соединении), PLIP (Parallel Line Internet Protocol), PPP (Point-to-Point Protocol), NFS (Network File System), и т.д. Поддерживается весь спектр клиентов и услуг TCP/IP, таких как FTP, telnet, NNTP и SMTP.

Ядро Linux сразу создано с учетом специального защищенного режима для процессоров Intel 80386 и выше. Любой, знакомый с защищенным режимом процессора 80386, знает, что этот чип проектировался для многозадачных систем, вроде UNIX (или Mulics).

Ядро Linux поддерживает загрузку только нужных страниц. То есть с диска в память загружаются те сегменты программы, которые действительно используются. Возможно использование одной страницы, физически один раз загруженной в память, несколькими выполняемыми программами.

Для увеличения объема доступной памяти Linux осуществляет также разбиение диска на страницы: то есть на диске может быть выделено до 256 Мбайт "пространства для свопинга" (swap space). Когда системе нужно больше физической памяти, то она с помощью свопинга выводит неактивные страницы на диск. Это позволяет выполнять более объемные программы и обслуживать одновременно больше пользователей.

Выполняемые программы используют динамически связываемые библиотеки, т.е. выполняемые программы могут совместно использовать библиотечную программу, представленную одним физическим файлом на диске. Это позволяет выполняемым файлам занимать меньше места на диске, особенно тем, которые многократно используют библиотечные функции. Есть также статические связываемые библиотеки для тех, кто желает пользоваться отладкой на уровне объектных кодов или иметь "полные" выполняемые программы, которые не нуждаются в разделяемых библиотеках.

Для обеспечения отладки ядро Linux выдает дампы памяти для "посмертного" анализа. Использование дампа и динамических отладчиков позволяет определить причины краха программы.

2.2. Программные характеристики

Практически любая утилита, которую вы ожидаете найти в стандартных реализациях UNIX, имеется и в Linux. Сюда включены и базовые команды, такие как ls, awk, tr, sed, bc, more и т.д. Многие пользователи самой важной утилитой считают Shell (программу, которая читает и выполняет команды пользователя). Кроме того, многие оболочки (shells) имеют такие возможности, как контроль выполнения (job control) (позволяя пользователю управлять несколькими параллельными процессами), перенаправление входа-выхода и командный язык для написания командных файлов (shell scripts). Командный файл - это программа на языке оболочки, аналогичная "batch file" в MS-DOS.

В Linux много типов оболочек. Наиболее важное различие между ними - используемый командный язык. Например, C Shell (csh) использует командный язык, чем-то напоминающий язык программирования Си. Классический баурновский Shell (Bourne Shell) использует иной командный язык.

2.3. Командная строка

ОС Linux, как и другие UNIX-подобные системы, ориентирована на управление с командной строки. В большинстве ваших исследований мира Linux вы будете общаться с ним через оболочку Shell. Shell - это просто программа, которая воспринимает введенное пользователем (т.е. команды, которые вы напечатаете) и транслирует это в команды системе. Это можно сравнить с программой COMMAND.COM под MS-DOS, которая делает нечто похожее. Shell - это лишь один из интерфейсов Linux. Существует много различных интерфейсов, таких как X Window System, который позволяет выполнять команды, используя мышь и клавиатуру в сочетании.

Как только вы вошли в систему под вашим паролем, она запускает Shell и вы можете вводить команды.

Ниже приведен список базовых команд Linux, необходимых для выполнения практического задания. Все имена файлов и команд чувствительны к большим и малым буквам (чего нет в системах типа MS-DOS). Например, команда make очень отличается от Make или MAKE. То же относится и к именам каталогов. Большинство команд имеют опции, которые позволяют уточнять действия, выполняемые командой.

cat

Используется для конкатенации файлов, а также для выдачи полного содержания файла разом.

Синтаксис: cat <file1> ... <fileN>,
где <file1> ... <fileN> - выдаваемые файлы.
Пример: cat letters/from-mdw выдает на дисплей файл letters/from-mdw.

cd

Изменяет текущий рабочий каталог.
Синтаксис: cd <directory>,
где <directory> - каталог, в который надо перейти (“.” ссылается на текущий каталог, “..” - на родительский каталог).

cp

Копирует файл(ы) в файл или каталог.
Синтаксис: cp <file1> ... <fileN> <destination>,
где <file1> ... <fileN> - имена копируемых файлов, а <destination> файл или каталог, в который копируют.
Пример: cp ../frog joe копирует файл ../frog в файл или каталог joe.

echo

Просто повторяет аргументы.
Синтаксис: echo <arg1> ... <argN>,
где <arg1> ... <argN> - "повторяемые" аргументы.
Пример: echo "Hello world" выдает на экран ``Hello world".

find

Осуществляет поиск файлов и каталогов.
Синтаксис: find <file1> ... <fileN> <destination>,
где <file1> ... <fileN> - искомые файлы или каталоги, а <destination> – имя каталога, в котором идет поиск. Если оно не указано, то поиск идет в текущем каталоге. В именах искомых файлов и каталогов можно использовать метаопределения “*” и “?”.
Пример: find test* /home/larry – найти все файлы, начинающиеся с test в каталоге /home/larry .

grep

Выдает все строки в названном файле(ах), которые содержат заданный образец.
Синтаксис: grep <pattern> <file1> ... <fileN>,
где <pattern> - образец (представленный регулярным выражением) и <file1> ... <fileN> - файлы, в которых производится поиск.
Пример: grep loomer /etc/hosts выдаст все строки, в которых файл /etc/hosts содержит образец “loomer”.

ls

Выдает информацию о файлах в каталоге.
Синтаксис: ls <file1> ... <fileN>,

где <file1> ... <fileN> - имена файлов или каталогов, информацию о которых надо выдать.

Наиболее часто используемые опции: -F (для представления информации о типах файлов) и -l (выдает в длинном формате информацию о размерах файлов, владельцах, правах доступа и т.д.)

Пример: ls -lF /home/larry выдаст содержимое каталога /home/larry.

man

Выдает страницу Руководства по данной команде или ресурсу. (здесь "ресурс" - это любая системная утилита, которая не является командой, например библиотечная функция).

Синтаксис: man <command>,

где <command> - имя команды или ресурса, о котором запрашивается информация.

Пример: man ls дает помощь по команде ls.

mkdir

Создает новые каталоги.

Синтаксис: mkdir <dir1> ... <dirN>,

где <dir1> ... <dirN> - создаваемые каталоги.

Пример: mkdir /home/larry/test создает каталог test в каталоге /home/larry.

more

Выдает содержимое названных файлов поэкранно.

Синтаксис: more <file1> ... <fileN>,

где <file1> ... <fileN> - отображаемые файлы.

Пример: more papers/history-final представляет файл papers/history-final.

mv

Перемещает файл(ы) в другой файл или каталог. Эта команда не эквивалентна копированию с последующим уничтожением оригинала. Она может быть использована для переименования файлов, как команда RENAME из MS-DOS.

Синтаксис: mv <file1> ... <fileN> <destination>,

где <file1> ... <fileN> - имена перемещаемых файлов, а <destination> имя файла или каталога, в который перемещают.

Пример: mv ../frog joe перемещает файл ../frog в файл или каталог joe.

rm

Удаляет файлы. Имейте в виду, когда в UNIX удаляются файлы, они невосстановимы.

Синтаксис: rm <file1> ... <fileN>,

где <file1> ... <fileN> - имена удаляемых файлов.

Опция -i потребует вашего подтверждения перед удалением файла. Пример: rm -i /home/larry/joe /home/larry/frog удаляет файлы joe и frog в каталоге /home/larry.

rmdir

Эта команда удаляет пустые каталоги. При использовании rmdir ваш текущий рабочий каталог должен находиться вне удаляемого каталога. Синтаксис: rmdir <dir1> ... <dirN>,

где <dir1> ... <dirN> удаляемые каталоги.

Пример: rmdir /home/larry/papers удаляет каталог /home/larry/papers, если он пустой.

3. Практическое задание

1. Вывести на экран содержимое каталога /home , не перемещаясь в него. При этом использовать опцию, позволяющую получить информацию о типах файлов.
2. Создать в своем домашнем каталоге подкаталог temp.
3. Скопировать файл devices из каталога /tmp/etc в свой домашний каталог.
4. Переместить файл devices из домашнего каталога в подкаталог temp.
5. Перейти в подкаталог temp и вывести содержимое указанного файла поэкранно.
6. Удалить файл из каталога temp.
7. Удалить каталог temp.

4. Вопросы для самоконтроля

1. Что такое Shell?
2. Что такое опция в команде и с какого символа она начинается?
3. Какие опции имеет команда ls и какие команда rm?
4. При помощи какой команды можно создать новый каталог и какой командой можно удалить каталог?
5. Что может делать команда mv кроме перемещения файлов?
6. Опишите синтаксис команды cp.
7. Можно ли восстановить файлы после удаления?
8. Опишите назначение и синтаксис команды cat.
9. Для чего нужна команда find?

Практическая работа № 2

ИССЛЕДОВАНИЕ ФАЙЛОВОЙ СИСТЕМЫ UNIX

1. Цель работы

Получение знаний о строении файловой системы Linux.

2. Теоретические сведения

Файловая система – это собрание файлов и иерархия каталогов. Если вы выполните команды cd / и ls -F , то, очевидно, увидите каталоги: bin, dev, etc, home, install, lib, mnt, proc, root, tmp, user, usr и var. (Можете увидеть и несколько отличный вариант, т.к. различные версии Linux имеют отличия).

Файлы рассортированы по этим каталогам в соответствии с выполняемыми ими функциями. Уточним, какое же назначение имеют эти каталоги.

/bin

bin (сокращенно от "binaries") - двоичные или выполняемые файлы. Здесь находится много важных системных программ. Используйте команду ls -F /bin, чтобы посмотреть имеющийся здесь список файлов. Вы можете обнаружить уже знакомые вам команды, вроде cp, ls и mv. Это и есть программы соответствующих команд. Когда, например, вы используете команду cp, вы выполняете программу /bin/cp.

Используя ls -F, вы увидите, что большинство файлов (если не все) в /bin имеют справа от имени звездочку “*”. Это говорит о том, что файлы выполняемые.

/dev

Следующая остановка на нашем пути - dev. Вновь посмотрите на содержимое с помощью ls -F.

Файлы в /dev известны как драйверы устройств, они используются для доступа к устройствам и ресурсам системы, таким как диски, модемы, память и т.д. Например, как вы читаете данные из файла, точно так же вы можете читать входные сигналы от мыши, имея доступ к /dev/mouse.

Имена файлов, начинающиеся на fd, - это дисководы гибких дисков (fd0 - первый дисковод, fd1 – второй).

Вот перечень некоторых из наиболее используемых файлов устройств.

- /dev/console/ относится к системной консоли, т.е. к монитору, напрямую связанному с системой.
- Различные /dev/ttyS и /dev/cua устройства используются для доступа к последовательным портам. Например, /dev/ttyS0 относится к "COM1" в MS-DOS. Устройства /dev/cua относятся к "звонящим" ("callout") устройствам, которые используются совместно с модемами.
- Устройства, имена которых начинаются с hd, имеют доступ к жестким дискам. /dev/hda относится ко *всему* первому жесткому диску, а hda1 - только к *первому разделу* /dev/hda.
- Устройства с именами, начинающимися на sd - SCSI-драйверы. Если у вас SCSI-жесткий диск, вместо доступа к нему через /dev/hda, вы будете обращаться к /dev/sda. SCSI ленты доступны через устройства st, а SCSI CD-ROM - через sr.
- Устройства lp обеспечивают доступ к параллельным портам. /dev/lp0 относится к "LPT1" в MS-DOS.
- /dev/null используется как "черная дыра" - любые данные, посланные сюда, канут в лету. Если вы хотите подавить вывод команды на экран, то можете перенаправить этот вывод в /dev/null.

/etc

/etc содержит множество всевозможных системных файлов конфигурации. Они включают /etc/passwd (файл паролей), /etc/rc (командный файл инициализации) и т.д.

/sbin

/sbin используется для хранения важных системных двоичных файлов, используемых системным администратором.

/home

home содержит домашние каталоги пользователей. Например, /home/larry - домашний каталог пользователя “larry”. На вновь инстал-

лированной системе этот каталог может быть пуст в связи с временным отсутствием зарегистрированных пользователей.

/lib

/lib содержит образы разделяемых библиотек (shared library images). Эти файлы содержат код, который могут использовать многие программы. Вместо того, чтобы каждая программа имела свою собственную копию этих исполняемых файлов, они хранятся в одном общедоступном месте - в /lib. Это делает исполняемые файлы меньше и экономит место в системе.

/proc

/proc - это "виртуальная файловая система", в которой файлы хранятся в памяти, а не на диске. Они связаны с различными процессами, происходящими в системе, и позволяют получить информацию о том, что делают программы и процессы в указанное время.

/tmp

Многие программы нуждаются в создании рабочих файлов, которые нужны короткое время. Каноническое место для этих файлов в /tmp (там обычно чаще проводится уборка мусора).

/usr

/usr - это очень важный каталог. Он состоит из ряда подкаталогов, которые, в свою очередь, содержат наиболее важные и полезные программы и файлы конфигурации, используемые системой.

Различные каталоги, описанные выше, необходимы для нормального функционирования системы, но большинство вещей, содержащихся в /usr, необязательны для системы. Но это такие необязательные вещи, которые делают систему полезной и интересной. Без /usr вы бы имели достаточно занудную систему, содержащую только программы, вроде cp и ls. /usr содержит много больших программных пакетов и конфигурационных файлов, которые их сопровождают.

/usr/bin

/usr/bin - настоящее хранилище для различных программ UNIX. Этот каталог содержит большинство выполняемых программ, которых нет ни в каких других местах, например в том же /bin их нет.

/usr/etc

Точно так же, как и /etc, содержит всевозможные системные программы и конфигурационные файлы и содержит большее количество утилит и файлов.

/usr/lib

/usr/lib содержит библиотеки-"заглушки" и "статические" библиотеки, эквивалентные файлам из /lib. При компиляции программа "связывается" с библиотеками, находящимися в /usr/lib, которые в свою очередь направляют программы обращаться в /lib, если им нужен актуальный код. Кроме того, многие другие программы хранят в /usr/lib свои конфигурационные файлы.

/usr/local

/usr/local в большой степени похож на /usr: он содержит различные программы и файлы, несущественные для системы, но превращающие ее в удовольствие и восторг. В общем, эти программы, находящиеся в /usr/local, специфичны для вашей системы, т.е. /usr/local сильно отличается в различных версиях UNIX. Здесь вы найдете такие большие программные пакеты, как TeX (система форматирования документов) и Emacs (большой и мощный редактор), если вы их установите.

/usr/man

Этот каталог содержит страницы Руководства. Здесь два подкаталога для каждого "раздела" Руководства. (С помощью команды "man man" вы можете получить более подробную информацию). Например, /usr/man/man1 содержит исходные тексты (неотформатированный оригинал) страниц Руководства в разделе 1, а /usr/man/cat1 содержит отформатированные страницы для раздела 1.

/usr/src

/usr/src содержит исходные коды для различных программ вашей системы. Наиболее важная вещь здесь - это /usr/src/linux, содержащий исходные коды ядра Linux.

/var

/var содержит каталоги, которые часто меняются в размере или имеют тенденцию быстро расти. К числу таких каталогов относятся:

/var/adm

/var/adm содержит различные файлы, интересные системному администратору, специфические системные файлы, фиксирующие ошибки и проблемы, возникающие в системе. Другие файлы фиксируют входы в систему, как и неудачные попытки войти.

/var/spool

/var/spool содержит файлы, которые предварительно формируются для других программ. Например, если ваша машина подключена к сети, входная почта будет помещаться в /var/spool/mail до тех пор, пока вы не прочитаете ее или не удалите. Входящие и исходящие новости помещаются в /var/spool/news и т.д.

3. Практическое задание

1. Составить список всех исполнимых файлов в каталоге /bin.
2. Вывести на экран все драйверы 1-го раздела 1-го винчестера.
3. Составить список пользователей, имеющих свои каталоги в /home.

4. Вопросы для самоконтроля

1. Что такое файловая система?
2. Перечислите каталоги, в которых хранятся системные файлы.
3. Какие буквы имеют в названии драйверы гибких, жестких дисков и CD?
4. Где хранятся файлы пользователей?
5. Где хранятся временные файлы?
6. Что содержит каталог /usr?
7. Где хранится файл паролей?

Практическая работа № 3

ТЕКСТОВЫЙ РЕДАКТОР KWORD

1. Цель работы

Получение навыков в использовании основных возможностей текстового редактора KWord при создании простых документов.

2. Теоретические сведения

2.1. Основные сведения о текстовом редакторе KWord

Текстовый редактор KWord - это аналог редактора Word для Windows, позволяющий создавать разного рода тексты с использованием всех современных видов форматирования. С помощью KWord вы сможете быстро преобразовать обычный текст в сложный документ, содержащий рисунки, диаграммы, таблицы, указатели, оглавления, сноски и многое другое.

Для запуска KWord из Shell необходимо набрать команду kword. Если вы находитесь в системе X Windows, то щелкните по кнопке «запуск приложения» и выберите пункты «Офис» – «KWord». После того как программа запустится, на экране появится окно, представленное на рис. 1. Здесь вы выбираете, открыть ли уже имеющийся документ или создать новый, а если новый, то какой.

После того, как вы выберете открываемый документ или вид нового документа, на экране появится окно текстового редактора KWord, изображенное на рис. 2.

По умолчанию в окне присутствуют элементы: строка заголовка, строка меню, четыре панели инструментов, вертикальная полоса прокрутки, кнопка системного меню, кнопки открытия и закрытия окна. В окне вы увидите вертикальный мигающий штрих - курсор.

При нажатии кнопки системного меню (в левом углу строки заголовка) можно выбрать размеры окна KWord и управлять им: можно переместить данное окно, изменить его размер или закрыть его. С помощью кнопок, расположенных в правом углу строки заголовка, можно свернуть окно

KWord, перевести его в полноэкранный режим или оконный режим и закрыть.

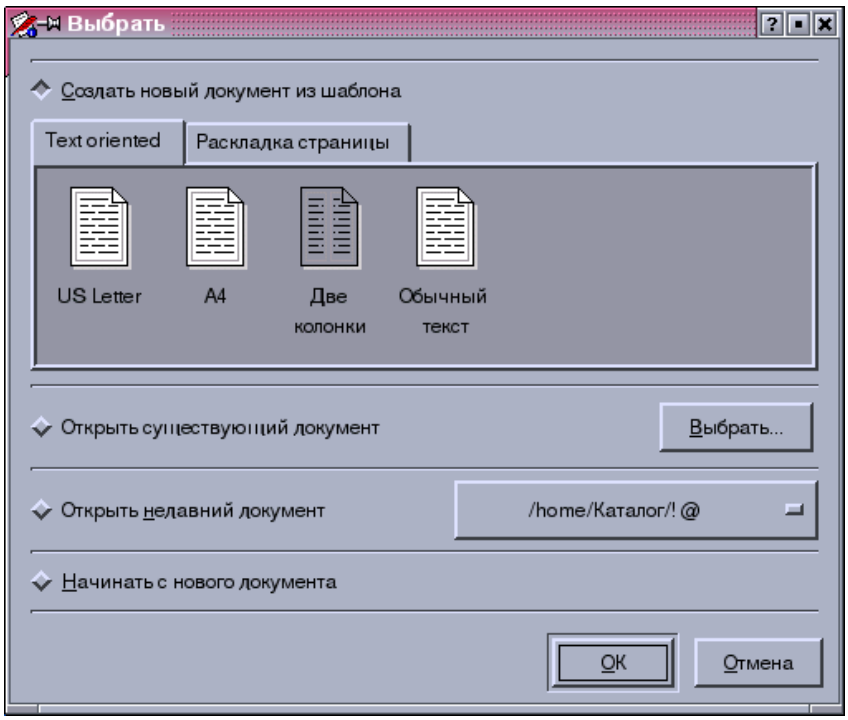

Рис. 1. Первое диалоговое окно при запуске KWord

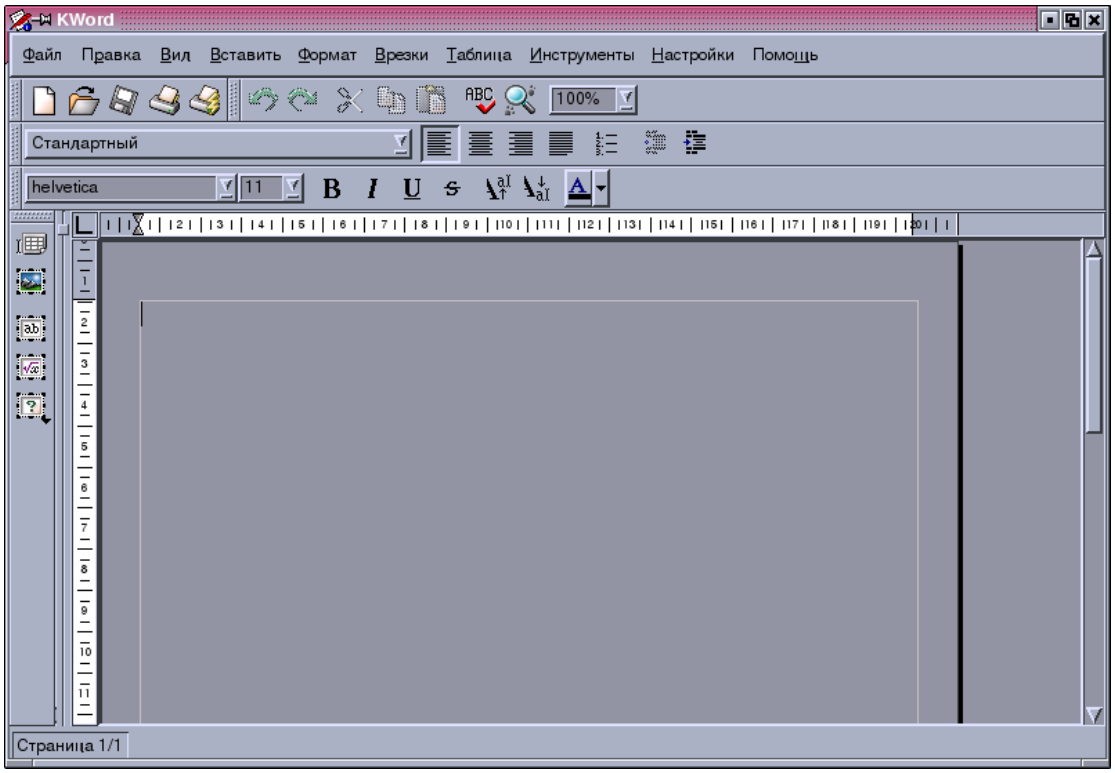

Рис. 2. Окно программы KWord

Главное меню. Главное меню расположено непосредственно под строкой заголовка KWord. В нем скрыты десять выпадающих меню с основными командами, необходимыми для организации управления и работы в текстовом редакторе. Если необходимо выполнить команду, вы открываете нужное меню и выбираете команду. Сделать это можно, щелкнув мышью на команде или набрав комбинацию клавиш. Любое меню можно вызвать комбинацией Alt+ подчеркнутая буква меню (меню «Формат» - Alt+ Ф). При перемещении курсора по выпавшему подменю команды выделяются подсветкой. Стрелка справа от команды указывает на то, что при выборе этого пункта появиться каскадное меню.

Панели инструментов. Для более быстрого и удобного способа доступа к командам редактора KWord предназначены кнопки на панелях инструментов. Они расположены ниже главного меню и дублируют его команды. Если задержать курсор мыши на кнопке, то появляется подсказка относительно функции данной кнопки. После запуска KWord по умолчанию показаны четыре панели: панель меню «Файл», панель меню «Правка», панель команды «Абзац» и панель команды «Шрифт». Однако существуют и другие панели. Вызвать дополнительные панели можно из меню «Настройки».

На панели меню «Файл» представлены кнопки: «создание нового документа», «открытие существующего документа», «сохранение открытого документа», «печать открытого документа», «предварительный просмотр перед печатью».

Панель меню «Правка» представлена кнопками для отмены действия, возврата отмененного действия, удаления в буфер, копирования в буфер и вставки из буфера, а также для проверки орфографии, поиска фрагмента текста и управления масштабом.

Панель команды «Абзац» расположена в третьей строке и позволяет выбрать один из стилей оформления текста, установить разные варианты выравнивания (по левом краю, по центру, по правому краю, по ширине), создать список и др.

Панель команды «Шрифт» расположена в четвертой строке и дает возможность управлять гарнитурой и размером шрифта, устанавливать полужирный, наклонный, подчеркнутый или зачеркнутый шрифт, делать нижний и верхний индексы и менять цвет текста.

Перемещение по документу. Справа от окна документа располагается полоса вертикальной прокрутки текста, а снизу - горизонтальной. Они появляются в тех случаях, когда текст вашего документа не помещается на

экране и требует прокрутки. Это достигается щелчком мыши на стрелках в полосах прокрутки. На полосе прокрутки находится бегунок, показывающий текущее положение окна относительно документа. Можно переместиться сразу на экран вверх или вниз, щелкнув на полосе прокрутки с соответствующей стороны бегунка. Такой же результат можно получить при нажатии клавиш управления PageUp - для перемещения на один экран вверх, PageDown - на экран вниз.

2.2. Работа с простыми текстами.

Создание, сохранение и открытие документов. Для создания нового документа можно воспользоваться командой «Создать». Для перемещения на новую строку используйте клавишу Enter, если ваш текст приблизился к правому краю листа, то переход на новую строку происходит автоматически. Для сохранения документа надо выбрать команду «Сохранить» из меню «Файл» или щелкнуть на соответствующей кнопке панели инструментов.

Для сохранения документа под другим именем надо выбрать команду «Сохранить как». При этом, а также когда вы сохраняете документ, который еще никак не назван, вы увидите окно (рис. 3), в котором вы можете выбрать, в какой каталог надо поместить файл (по умолчанию – в домашний каталог), а также в каком формате надо сохранить документ: в большинстве случаев (и установленный по умолчанию) это формат KWord.

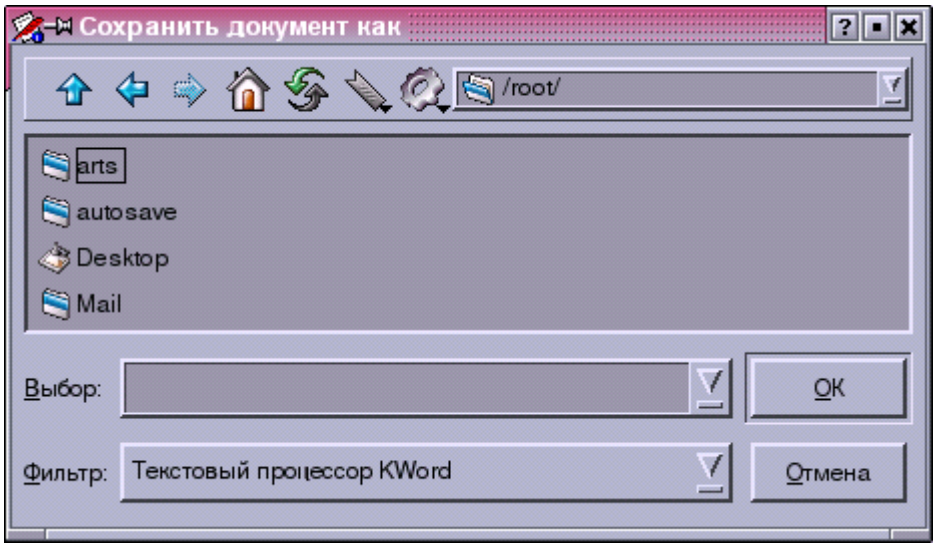

Рис. 3. Диалоговое окно сохранения документа

Для открытия документа надо выбрать команду «Открыть» из меню «Файл». Вы увидите следующее окно (рис. 4), в котором надо выбрать каталог, где находится документ, и сам документ.

Работа с текстом: удаление, выделение, копирование, перемещение. Для того чтобы скопировать текст или изменить шрифт, текст необходимо сначала выделить. При этом выделенный фрагмент окрашивается в черный цвет.

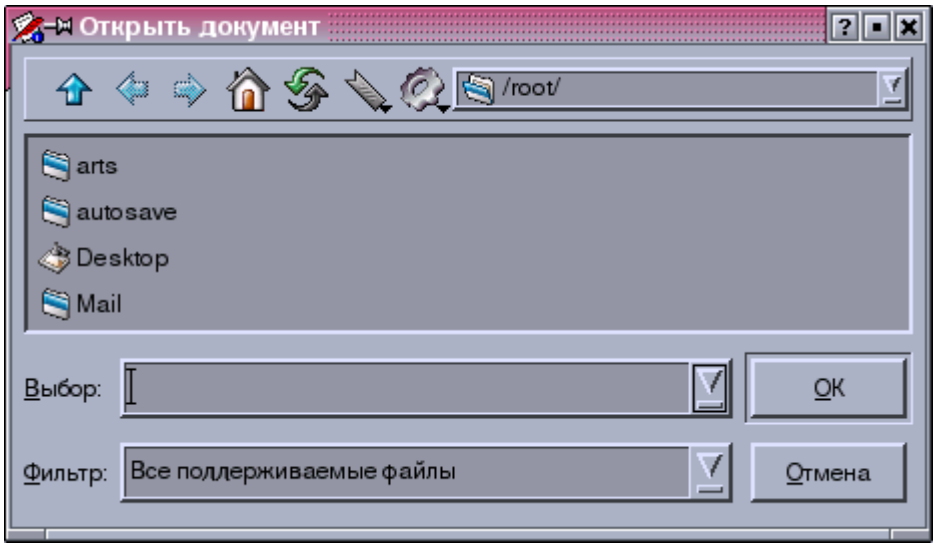

Рис. 4. Диалоговое окно открытия документа

В данной таблице представлены основные способы выделения текстовых блоков.

Фрагмент	**Способ выделения**
Любой	Переместить указатель мыши с нажатой левой кнопкой по тексту, который нужно выделить. Удерживая клавишу Shift, щелкнуть в начале и конце выделения. Удерживая клавишу Shift, нажать на клавиши управления курсором.
Строки	Поместить указатель мыши слева от строк и при изменении его вида на белую стрелку потянуть мышь вниз. Установить курсор в начало строки и нажать Shift+ End. Установить курсор в конец строки и нажать Shift+ Home
Слово	Дважды щелкнуть на слове.
Многостраничный фрагмент	Удерживая клавишу Shift, нажать на клавиши PageUp, PageDown.
Весь текст	Меню «Правка», команда «Выделить все».

Для отмены выделения надо щелкнуть мышью где-нибудь внутри или вне выделенного фрагмента, в любом месте окна.

Выделенный фрагмент можно скопировать в буфер обмена при нажатии кнопки «Копировать», или Ctrl+C.

Выделенный фрагмент можно переместить в буфер обмена при нажатии кнопки «Вырезать», или Ctrl+X. При этом текст исчезнет с экрана, но не пропадет. Его можно будет вставить в новое место, новый документ и даже в другую программу.

После того как вы поместили текст в буфер, его можно вставить в новое место. Для этого надо поставить указатель в нужное место, нажать кнопку «Вставить», или Ctrl+V. После того, как текст будет помещен на новое место, он по прежнему останется в буфере обмена. Поэтому чтобы текст вставить несколько раз, его достаточно только один раз поместить в буфер. Текст останется в буфере до тех пор, пока туда не будет помещен другой фрагмент.

Метод «перенести и оставить». Если вы выделили часть текста, то ее можно быстро переставить в новое место, при этом не используя буфер обмена. Для этого после выделения надо «взять» фрагмент мышью и переместить в новое место. При этом вертикальная пунктирная полоса покажет место новой вставки выделенного фрагмента. При выполнении данной операции с нажатой клавишей Ctrl фрагмент будет скопирован в новое место, а не перемещен.

При наборе текста могут возникнуть ошибки. Если вы неправильно набрали символ, то его можно стереть нажатием BackSpace или Delete: BackSpace стирает символ слева от курсора, а Delete – справа. Неправильно набранный текст можно отменить кнопкой «Отменить» или комбинацией Ctrl+Z. Клавиша Insert включает режим вставки и замены. В режиме вставки при нажатии клавиши KWord вставляет символ слева от курсора. В режиме замены следующий за курсором символ заменяется на новый.

Поиск и замена фрагментов текста. Для этого необходимо воспользоваться командой «Поиск» или командой «Замена» из меню «Правка». Диалоговое окно, вызываемое командой «Замена», представлено на рис. 5. В нем вы можете указать искомый фрагмент, фрагмент, на который вам надо заменить искомый, а также установить различные параметры поиска, такие как учет регистра, поиск полных слов или полных слов и фрагментов. Также можно осуществлять поиск с начала текста или с позиции кур-

сора, в прямом (сверху вниз по тексту) или обратном (снизу вверх) порядке, или только в выделенном фрагменте текста.

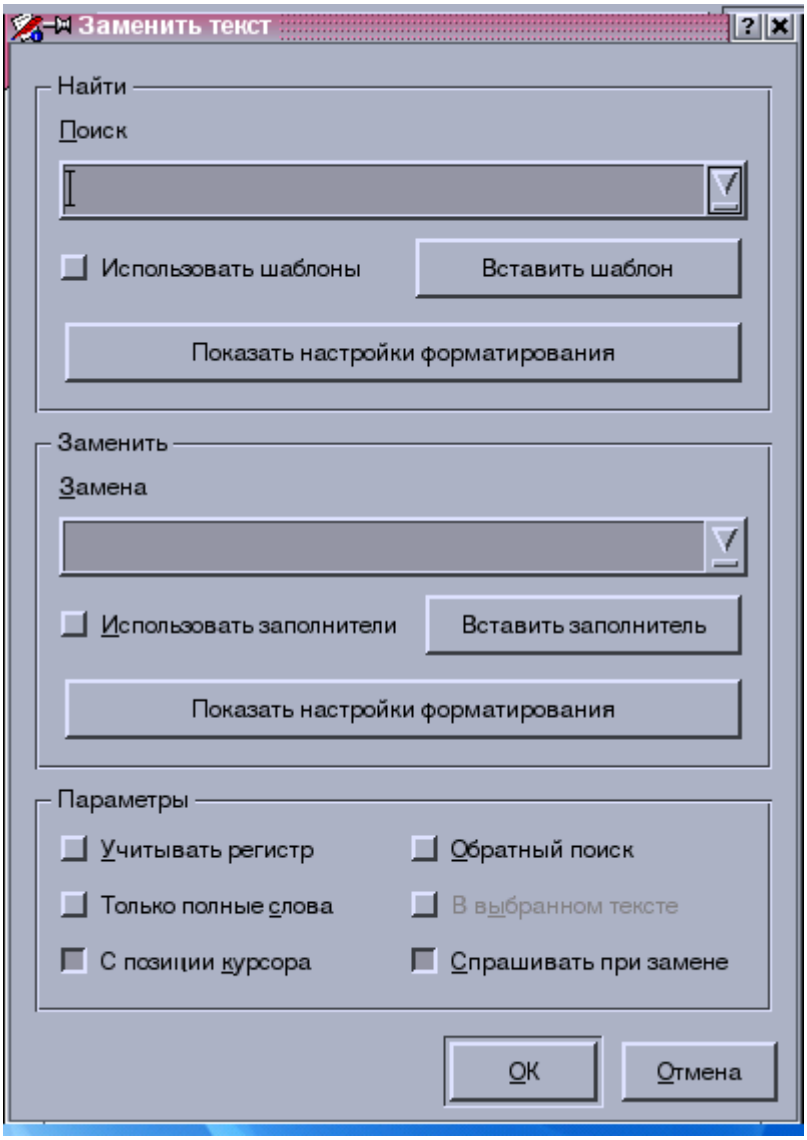

Рис. 5. Диалоговое окно замены текста

Шрифтовое оформление. KWord предоставляет возможности установки различной гарнитуры (вида символов), размера и других параметров шрифта. Для этого необходимо выделить фрагмент текста, шрифт которого вы хотите изменить, и выбрать команду «Шрифт» из меню «Формат».

В окне этой команды (рис. 6) можно выбрать различные параметры шрифта: гарнитуру, стиль (обычный, курсив, полужирный), размер, кодировку (Windows 1251, koi8-r), цвет шрифта и оформление символов (можно сделать текст в виде индексов, подчеркнутый, зачеркнутый и т.д.).

То же самое можно сделать, используя соответствующие кнопки панелей инструментов.

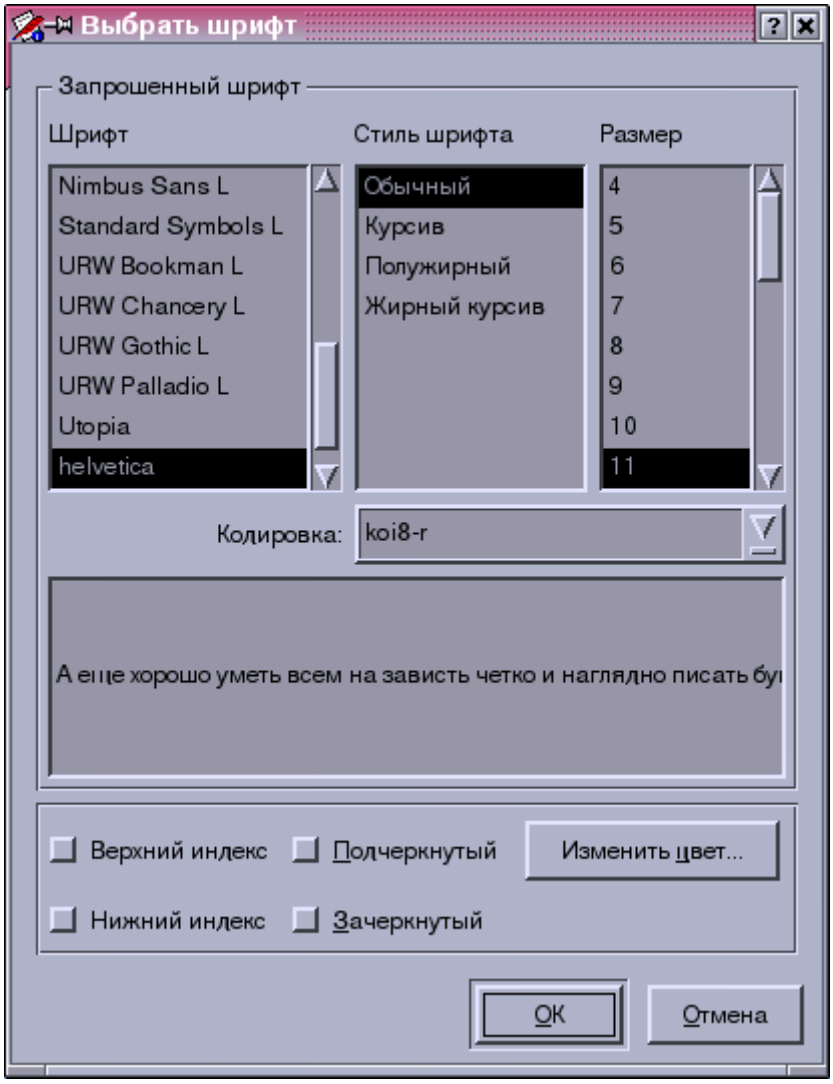

Рис. 6. Диалоговое окно команды «Шрифт»

Для вставки в текст символов, которых нет на клавиатуре, существует команда «Специальный символ» из меню «Вставить». В ее диалоговом окне (рис. 7) можно выбрать символ из нескольких специальных таблиц, а также установить гарнитуру и другие параметры шрифта этого символа.

В таблицах имеется очень широкий набор символов. Это буквы греческого и других алфавитов, знаки транскрипции английского языка, математические символы, знаки, используемые в экономике, знаки, которые можно применять для оформления официальных и неофициальных писем и пр.

Кроме того, в этих таблицах присутствуют и все те символы, которые есть на клавиатуре.

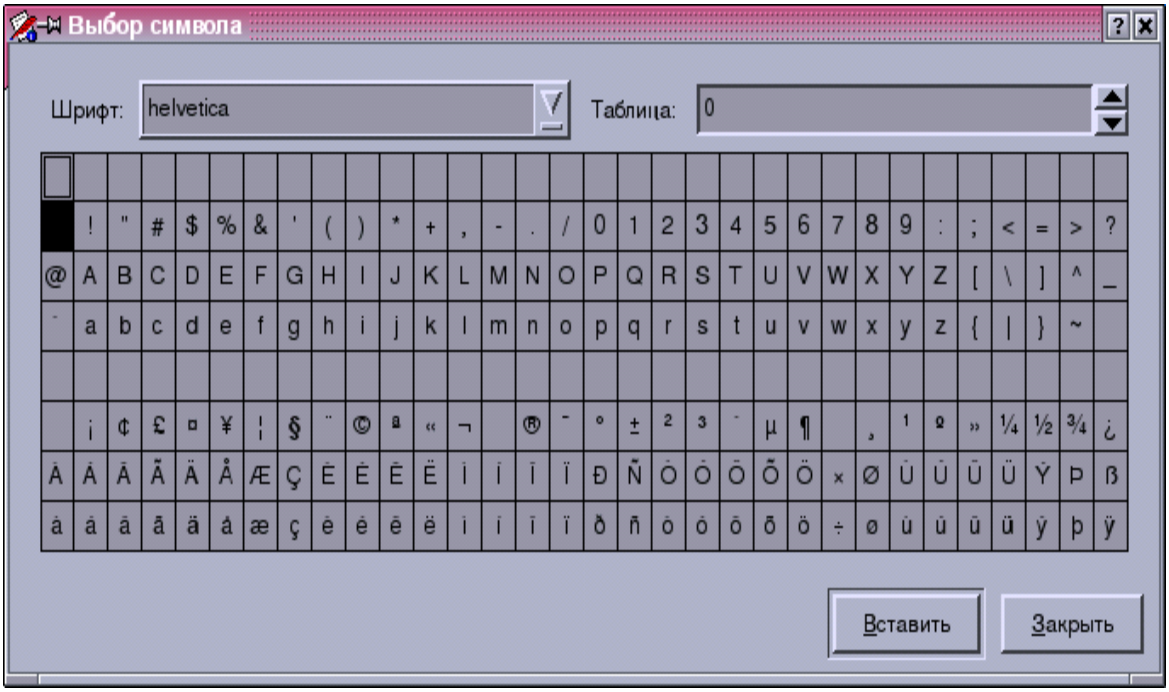

Рис. 7. Диалоговое окно команды «Специальный символ»

2.3. Форматирование абзацев и страниц

Выравнивание абзацев и заголовков. Способы выравнивания абзацев и заголовков определяют, каким образом набранный текст располагается относительно правого и левого края страницы. Для изменения этих параметров можно использовать кнопки панели форматирования или воспользоваться вкладкой «Выравнивание» диалогового окна «Свойства абзаца», вызываемого командой «Абзац» из меню «Формат».

Отступы и интервалы. Для установления отступов и интервалов необходимо воспользоваться соответствующей вкладкой окна «Свойства абзаца» (рис. 8). Здесь можно установить отступы текста справа и слева, отступ красной строки, параметры переноса абзацев и межстрочный интервал.

Создание списков. Для этого необходимо воспользоваться вкладкой «Точки/Числа» того же диалогового окна. Здесь можно выбрать вид нумерации или маркировки пунктов списка: арабские, римские цифры, различ-

ного вида маркеры. В качестве маркера, вообще говоря, можно использовать любой символ из таблицы символов. Для этого нужно щелкнуть по кнопке «-» справа от слов «пользовательский символ» (рис. 9).

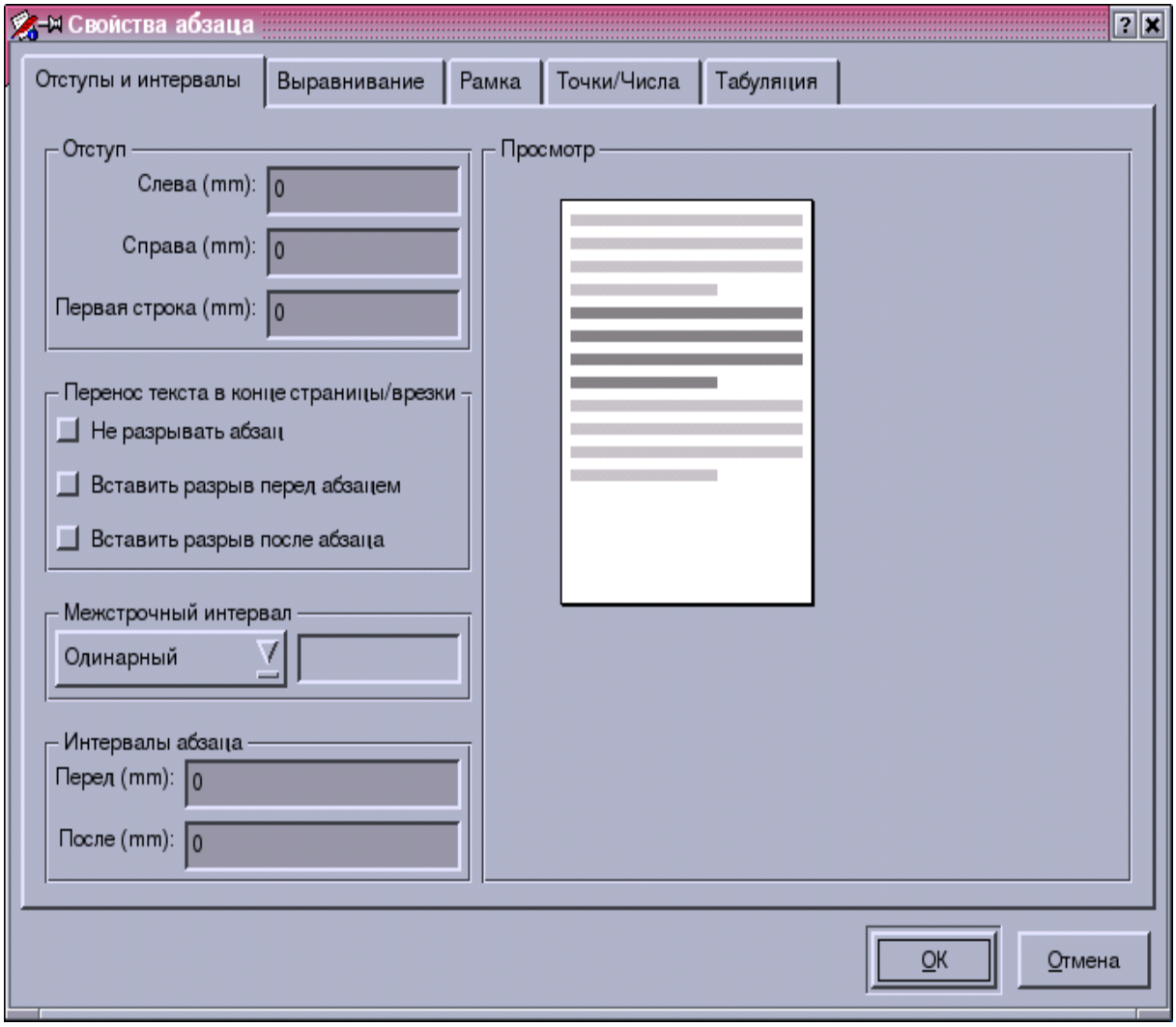

Рис. 8. Вкладка “Отступы и интервалы” команды “Абзац”

Параметры страницы. Эти параметры устанавливаются командой «Страница» из меню «Формат» (рис. 10).

Параметры страницы – это, прежде всего, формат бумаги (А4, А3, т.е. все часто используемые форматы и ряд других; можно определить произвольный размер бумаги), ориентация страницы (книжная или альбомная), размеры полей. Также на страницах могут размещаться колонтитулы (верхние и нижние), их параметры можно задать на одноименной вкладке.

Все изменения, вносимые пользователем в параметры страницы, незамедлительно находят свое отражение на образце, расположенном в этом диалоговом окне справа (см. рис. 10).

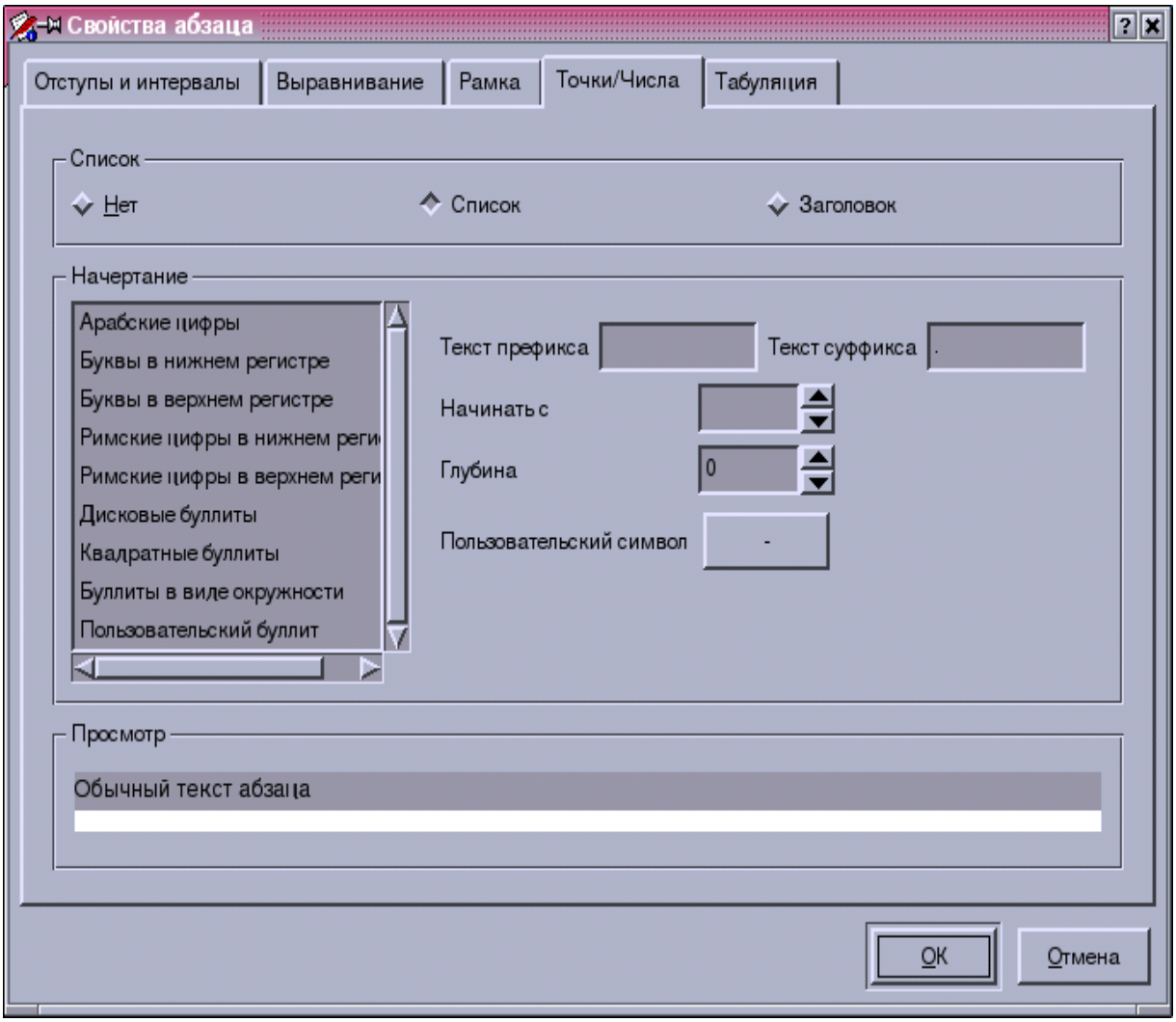

Рис. 9. Вкладка “Точки/Числа” команды “Абзац”

Кроме перечисленного выше с помощью текстового редактора KWord можно редактируемый текст окружить рамкой. Пользователю предоставляется возможность выбрать вид рамки из списка стандартных (при этом линии сверху, снизу, справа и слева будут одинаковыми) или для каждой стороны установить свой тип линии. Можно изменять как тип, так и толщину линии. Для этого необходимо перейти на вкладку «Рамка» команды «Абзац» из меню «Формат».

Текст иногда должен начинаться не от границы левого поля бумаги. Можно расположить текст в несколько столбцов (не колонок, а именно

столбцов, т.е. строки текста будут заканчиваться у правой границы бумаги), выровняв их по табуляции. Вы можете увидеть пример такого расположения, если откроете оглавление данной книги. При этом заполнитель табуляции может состоять из пробелов, точек, сплошной линии и других символов. Определить границы табуляции и вид заполнителя можно на вкладке «Табуляция» той же команды «Абзац».

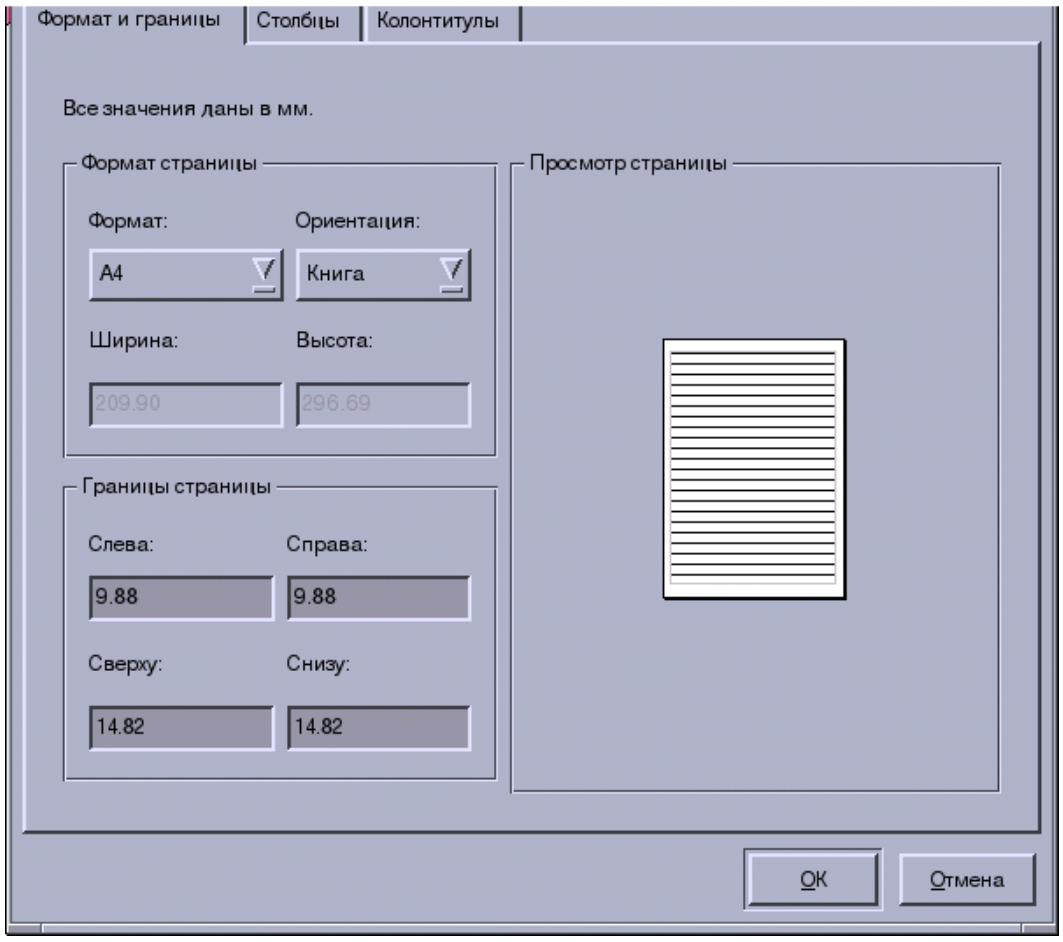

Рис. 10. Диалоговое окно команды “Страница”

2.4. Создание таблиц

Для создания новой таблицы необходимо воспользоваться командой «Таблица» из меню «Вставить». В открывшемся диалоговом окне можно установить количество строк и столбцов, высоту и ширину ячеек (рис. 11).

Для дальнейшей работы с отдельной ячейкой необходимо ее выделить (щелкнуть левой кнопкой мыши по линии границы). KWord предоставляет возможности добавить строки и столбцы в таблицу, удалить их, разбить ячейку на несколько ячеек, объединить группу ячеек. Все это доступно из меню «Таблица».

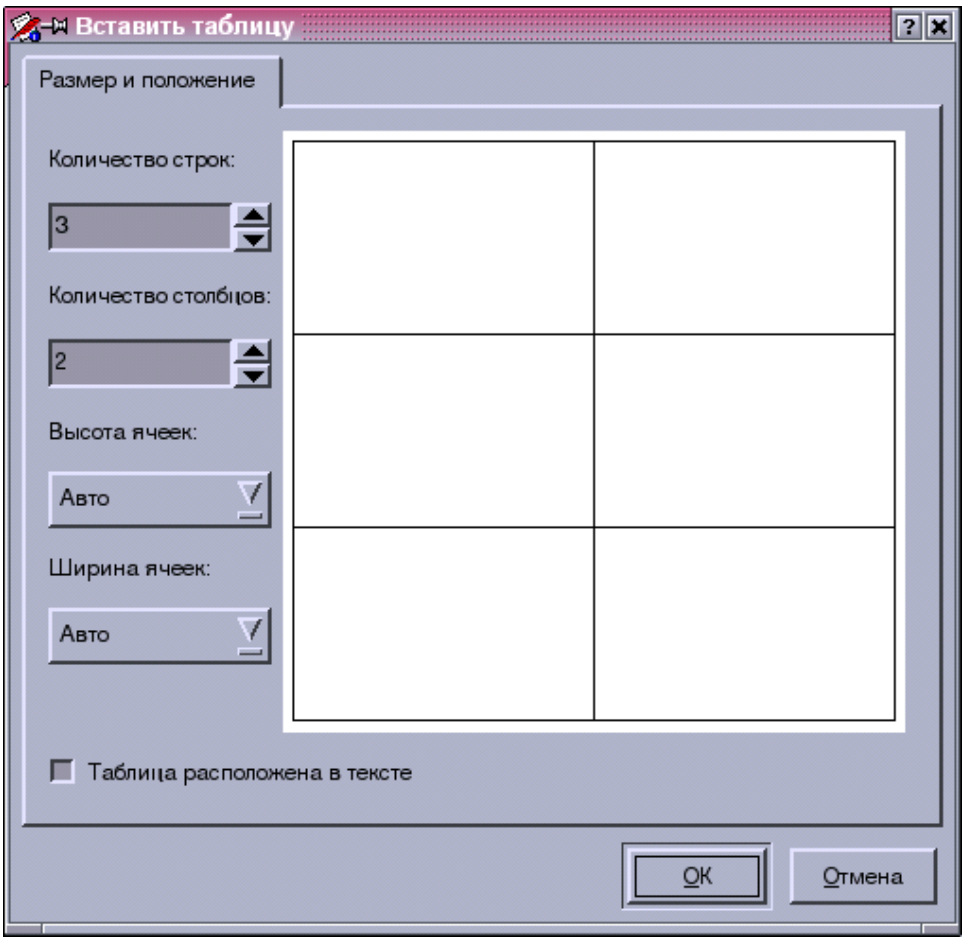

Рис. 11. Диалоговое окно команды «Таблица»

3. Практическое задание

1. Создать новый файл и сохранить его в своем домашнем каталоге.
2. Набрать любой текст, состоящий из двух абзацев по 5 - 7 строк в каждом и трех отдельных строк.
3. Установить для первого абзаца шрифт times, размер 13, подчеркнутый; выравнивание по ширине; отступ первой строки 1,5 см.

4. Установить для второго абзаца любой шрифт по желанию, размер 14, полужирный; выравнивание по правому краю. Второе слово написать в виде верхнего индекса (зачеркнутый).
5. Три отдельные строки оформить в виде списка с нумерацией римскими цифрами.
6. Между простым текстом и списком добавить таблицу из 5 строк и 4 столбцов. В таблице объединить все ячейки первой строки в одну, а также объединить ячейки второй строки 1-го и 2-го столбцов и ячейки той же строки 3-го и 4-го столбцов. Заполнить таблицу любыми данными.
7. Установить параметры страницы: альбомное расположение бумаги (А4), верхнее поле – 2,0 см, нижнее и левое – по 3,0 см, правое – 1,5 см. Добавить верхний колонтитул с любым текстом.

4. Вопросы для самоконтроля

1. Для чего предназначена программа KWord и как ее запустить?
2. Какие элементы содержит окно программы KWord?
3. Какие кнопки имеются в строке заголовка и для чего они предназначены?
4. Какие пункты содержит главное меню? Их назначение?
5. Как можно сохранить документ под другим именем и открыть уже имеющийся документ?
6. Какие существуют способы выделения текстовых блоков?
7. Как можно вырезать и скопировать текст в буфер, вставить текст из буфера, а также перемещать и копировать фрагменты текста без использования буфера?
8. Как можно найти и заменить фрагмент текста?
9. Какие возможности есть в программе KWord для шрифтового оформления текста?
10. Как можно вставить в текст символ, которого нет на клавиатуре (например букву греческого алфавита)?
11. Что такое выравнивание текста, какие бывают виды выравнивания и как их установить?
12. Что включается в параметры страницы и как их можно изменять?
13. Как можно добавить в текст таблицу и какие преобразования можно делать в таблице?

Практическая работа № 4

ПРОГРАММИРОВАНИЕ НА SHELL

1. Цель работы

Получение практических навыков в написании командных файлов.

2. Теоретические сведения

Командный язык Shell (в переводе - раковина, скорлупа) фактически есть язык программирования очень высокого уровня. На этом языке пользователь осуществляет управление компьютером. Обычно после входа в систему вы начинаете взаимодействовать с командной оболочкой.

Shell не является необходимым и единственным командным языком. Например, немалой популярностью пользуется язык cshell, есть также kshell, bashell (из наиболее популярных в последнее время) и другие.

Одна из многих команд UNIX - shell. То есть в набор команд оболочки (интерпретатора) Shell входит команда sh - вызов интерпретатора Shell. Первый Shell вызывается автоматически при вашем входе в систему. После этого вы можете вызывать на выполнение любые команды, в том числе и снова сам Shell, который вам создаст новую оболочку внутри прежней.

Стержневым элементом языка Shell является команда.

2.1. Структура команд

Команды в Shell обычно имеют следующий формат:

<имя команды> <флаги> <аргумент(ы)>

Например:

ls -ls /usr/bin

где ls - имя команды выдачи содержимого директория, -ls - флаги ("-" - признак флагов, l - длинный формат, s - объем файлов в блоках), /usr/bin - каталог, для которого выполняется команда.

Эта команда выдаст на экран в длинном формате содержимое каталога /usr/bin, при этом добавит информацию о размере каждого файла в блоках.

К сожалению, такая структура команды выдерживается далеко не всегда. Не всегда перед флагами ставится минус, не всегда флаги идут одним словом. Есть разнообразие и в представлении аргументов. К числу команд,

имеющих экзотические форматы, относятся и такие "ходовые" команды, как cc, tar, dd, find, и ряд других. Как правило (но не всегда), первое слово (т.е. последовательность символов до пробела, табуляции или конца строки) Shell воспринимает как команду.

2.2. Группировка команд

Средства группировки:
; и <перевод строки> - определяют последовательное выполнение команд;
& - асинхронное (фоновое) выполнение предшествующей команды;
&& - выполнение последующей команды при условии нормального завершения предыдущей, иначе игнорировать;
|| - выполнение последующей команды при ненормальном завершении предыдущей, иначе игнорировать.

Иногда необходимо, чтобы все фоновые процессы завершились прежде, чем будет выполняться какой-то расчет. Для этого служит специальная команда "wait [PID]". Эта команда ждет завершения указанного идентификатором (числом) фонового процесса. Если команда без параметра, то она ждет завершения всех фоновых процессов, дочерних для данного "sh".

Для группировки команд также могут использоваться фигурные "{}" и круглые "( )" скобки. Рассмотрим примеры, сочетающие различные способы группировки. Если введена командная строка

k1 && k2; k3

где "k1", "k2" и "k3" - какие-то команды, то "k2" будет выполнена только при успешном завершении "k1"; после любого из исходов обработки "k2" будет выполнена "k3".

k1 && {k2; k3}

Здесь обе команды ("k2" и "k3") будут выполнены только при успешном завершении "k1".

{k1; k2} & k3

В этом случае команда "k3" будет выполнена после успешного завершения обеих команд "k1" и "k2".

Для уничтожения фонового процесса надо знать его номер. При запуске фонового процесса на экран выдается число, соответствующее номеру (идентификатору) этого процесса (PID). Если этот номер забыт или надо убедиться, что процесс не закончен, с помощью команды ps -aux можно получить перечень идентификаторов процессов (PID), имена пользователей, текущее время, затраченное процессами, и т.д. В выведенной таблице можно найти номера процессов, подлежащих уничтожению, например это "849" и "866". Тогда командой

kill -9 866 849

можно уничтожить эти процессы. При уничтожении процессов вы должны иметь то же имя пользователя, какое было приписано уничтожаемым процессам (или иметь имя привилегированного пользователя).

Если параллельно обрабатывается или создается файл с *одним* именем (например, несколько пользователей вызвали в редактор один и тот же файл), то в системе продолжит существование тот вариант файла, который возвращен (записан) в систему последним.

Круглые скобки "( )" кроме выполнения функции группировки, выполняют функцию вызова нового экземпляра интерпретатора Shell. Пусть мы находились в начальном каталоге /mnt/lab/asu, тогда в последовательности команд

cd ..; ls; ls

две команды ls выдадут 2 экземпляра содержимого каталога /mnt/lab, а последовательность

(cd ..; ls) ls

выдаст сначала содержимое каталога /mnt/lab, а затем содержимое /mnt/lab/asu, т.к. при входе в скобки вызывается новый экземпляр Shell, в рамках которого и осуществляется переход. При выходе из круглых скобок происходит возврат в старый Shell и в старый каталог.

2.3. Перенаправление команд

Стандартный ввод (вход) stdin в ОС UNIX осуществляется с клавиатуры терминала, а стандартный вывод (выход) stdout направлен на экран терминала.

Команда, которая может работать со стандартным входом и выходом, называется *фильтром*. Пользователь имеет удобные средства перенаправления ввода и вывода на другие файлы (устройства). Символы ">" и ">>"обозначают перенаправление вывода.

ls > f1

Команда ls сформирует список файлов текущего каталога и поместит его в файл f1 (вместо выдачи на экран). Если файл f1 до этого существовал, то он будет стерт новым.

pwd >> f1

Команда pwd сформирует полное имя текущего каталога и поместит его в конец файла f1, т.е. ">>" добавляет в файл, если он непустой.

Символы "<" и "<<" обозначают перенаправление ввода.

Поскольку устройства в ОС UNIX представлены специальными файлами, их можно использовать при перенаправлениях. Специальные файлы

находятся в каталоге /dev. Например lp - печать; console - консоль; ttyi - i-ый терминал; null – фиктивный (пустой) файл (устройство).

Тогда, например,

ls > /dev/lp

выведет содержимое текущего каталога на печать, а

f1 < /dev/null

обнулит файл f1.

sort f1 | tee /dev/lp | tail –20

В этом случае будет отсортирован файл f1 и передан на печать, а 20 последних строк также будут выданы на экран. Вернемся к перенаправлению выхода. Стандартные файлы имеют номера: 0 - stdin, 1 - stdout и 2 - stderr. Если вам нежелательно иметь на экране сообщение об ошибке, вы можете перенаправить его с экрана в указанный вами файл (или вообще "выбросить", перенаправив в файл "пустого устройства" - /dev/null). Например, при выполнении команды

cat f1 f2

которая должна выдать на экран последовательно содержимое файлов f1 и f2, вы увидите на экране следующее

111111 222222

cat: f2: No such file or directory

где 111111 222222 - содержимое файла f1, а файл f2 отсутствует, о чем команда cat выдала сообщение в стандартный файл диагностики, по умолчанию, как и стандартный выход, представленный экраном. Если вам не желательно такое сообщение на экране, его можно перенаправить в указанный вами файл:

cat f1 f2 2>f-err

сообщения об ошибках будут направляться в файл f-err. Кстати, вы можете всю информацию направлять в один файл ff, использовав в данном случае конструкцию

cat f1 f2 >>ff 2>ff

Можно указать не только, какой из стандартных файлов перенаправлять, но и в какой стандартный файл осуществить перенаправление.

cat f1 f2 2>>ff 1>&2

Здесь сначала stderr перенаправляется (в режиме добавления) в файл ff, а затем стандартный выход перенаправляется на stderr, которым к этому моменту является файл ff. То есть результат будет аналогичен предыдущему. Конструкция 1>&2 означает, что кроме номера стандартного файла, в который надо перенаправить вывод, необходимо впереди ставить "&"; вся конструкция пишется без пробелов. <& закрывает стандартный ввод. >& закрывает стандартный вывод.

2.4. Генерация имен файлов

При генерации имен используют метасимволы: * - произвольная (возможно пустая) последовательность символов; ? - один произвольный символ; [...] - любой из символов, указанных в скобках перечислением и/или с указанием диапазона;

cat f* - выдаст все файлы каталога, начинающиеся с "f";

cat *f* - выдаст все файлы, содержащие "f";

cat program.? - выдаст файлы данного каталога с однобуквенными расширениями, скажем "program.c" и "program.o", но не выдаст "program.com";

cat [a-d]* - выдаст файлы, которые начинаются с "a", "b", "c", "d". Аналогичный эффект дадут и команды "cat [abcd]*" и "cat [bdac]*".

2.5. Командные файлы

Для того чтобы текстовый файл можно было использовать как команду, существуют несколько возможностей. Пусть с помощью редактора создан файл с именем "cmd".

Текстовый файл можно сделать выполняемым с помощью команды

chmod 755 cmd

Можно вызвать Shell как команду, обозначаемую sh, и передать ей файл cmd как аргумент или как перенаправленный вход, т.е. $ sh cmd или $ sh. Введенное вами число воспринимается интерпретатором не как число, а как последовательность символов. Он не проверяет, что вы ввели. Поэтому в качестве значения переменной может оказаться любая введенная абракадабра или просто нажатие как значение пустой строки. Для обеспечения проверки формата ввода следует написать свою команду. При обращении к Shell-переменной необходимо перед именем ставить символ "$". Так команды

echo $var_2
echo var_2

выдадут на экран

```
ОС UNIX
var_2
```

Там, где выполняется присваивание, пробелы *недопустимы*.

Присваивание, скажем, w= означает присваивание переменной "w" пустой строки. Но и пустую строку лучше присваивать аккуратно, напри-

мер w="". Для того, чтобы имя переменной не сливалось со строкой, следующей за именем переменной, используются фигурные скобки. Пусть

a=/mnt/lab/asu/

Тогда

cat /mnt/lab/asu/prim

и

cat ${a}prim

равноценны (т.е. "cat" выдаст на экран содержимое одного и того же файла). Если также предположить, что в системе есть переменная "prim" и "prim=dir" , то команда

echo ${a}$prim

выдаст на экран

/mnt/lab/asu/dir

2.5.1. Экранирование

Рассмотрим более подробно приемы экранирования, используемые в Shell. В качестве средств экранирования применяют двойные кавычки (" "), одинарные кавычки (' ') и бэк-слэш (\). Из примеров очевидно их действие - можно в одной строке записывать несколько присваиваний.

x=22 y=33 z=$x

A="$x" B='$x' C=\$x

D="$x + $y + $z" E='$x + $y + $z' F=$x\ +\ $y\ +\ $z

(присваивание G=$x + $y не было бы выполнено из-за пробелов). Тогда

echo A = $A B = $B C = $C

echo D = $D E = $E F = $F

eval echo evaluated A = $A

eval echo evaluated B = $B

eval echo evaluated C = $C

выдадут на экран:

A = 22 B = $x C = $x

D = 22 + 33 + 22 E = $x + $y + $z F = 22 + 33 + 22

evaluated A = 22

evaluated B = 22

evaluated C = 22

В трех последних случаях использована своеобразная команда eval, которая в подставленной в нее (в качестве аргумента) команде выполняет

означивание переменных (если таковые имеются). В результате значение A остается прежним, поскольку A имеет значение "22". А переменные B и C имеют значение "$x". За счет означивания, которое было выполнено командой eval B и C дают значения "22". Еще один пример с eval. Пусть

w=\$v v=\$u u=5

В результате выполнения команд

echo $w
eval echo $w
eval eval echo $w

на экран будет выведено:

$v
$u
5

Приведем еще примеры, связанные с экранированием перевода строки. Пусть переменной "string" присвоено значение массива размерностью 2 на 3 символа:

abc
def

Обратим внимание, что для избежания присваивания лишних пробелов вторая строка массива начата с первой позиции следующей строки:

string="abc
def"

Тогда три варианта записи переменной в команде "echo"

echo $string
echo '$string'
echo "$string"

дадут соответственно три различных результата:

abc def

$string

abc
def

Заметим также, что бэк-слэш (\) не только экранирует следующий за ним символ, что позволяет использовать специальные символы просто как символы, представляющие сами себя (он может экранировать и сам себя - \\), но в командном файле бэк-слэш позволяет объединять строки в одну (экранировать конец строки). Приводившийся ранее пример командной строки

cat f1 | grep -h result | sort | cat -b > f2

может быть записан в командном файле, скажем, как

**cat f1 | grep -h **
result | sort | cat -b > f2

Кстати, эффект продолжения командной строки обеспечивает и символ конвейера. В данном случае это может дать более симпатичный результат, например:

```
cat f1          |
grep -h result |
sort            |
cat -b > f2
```

2.5.2. Манипуляции с Shell-переменными

Несмотря на то, что Shell-переменные в общем случае воспринимаются как строки, т.е "35" - это не число, а строка из двух символов "3" и "5", в ряде случаев они могут интерпретироваться иначе, например, как целые числа. Разнообразные возможности имеет команда "expr". Проиллюстрируем некоторые на примерах. Выполнение командного файла:

```
x=7 y=2
a=`expr $x + $y` ;  echo a=$a
a=`expr $a + 1`  ;  echo a=$a
b=`expr $y - $x` ;  echo b=$b
c=`expr $x '*' $y`   ;  echo c=$c
d=`expr $x / $y` ;  echo d=$d
e=`expr $x % $y`   ;  echo e=$e
```

выдаст на экран

```
a=9
a=10
b=-5
c=14
d=3
e=1
```

Операция умножения "*" обязательно должна быть заэкранирована, поскольку в Shell этот значок воспринимается как спецсимвол, означающий, что на это место может быть подставлена любая последовательность символов. Следует обратить также внимание на обязательные пробелы, отделяющие переменные и знаки операций. С командой "expr" возможны не только (целочисленные) арифметические операции, но и строковые:

```
A=`expr 'cocktail' : 'cock'` ;  echo $A
B=`expr 'cocktail' : 'tail'`   ;  echo $B
C=`expr 'cocktail' : 'cook'` ;  echo $C
D=`expr 'cock' : 'cocktail'` ;  echo $D
```

На экран будут выведены числа, показывающие число совпадающих символов в цепочках (от начала). Вторая из строк не может быть длиннее первой:

```
4
0
0
0
```

И наконец, об условной замене переменных. Если переменные, скажем "x", "y", "z", не определены, то при обращении к переменным

${x-new} - в качестве значения "x" будет выдано "new",

${y=new} - в качестве значения "y" будет присвоено "new",

${z?new} - в качестве значения "z" будет выдано "z: new" и соответствующая процедура прекращается.

Во всех этих случаях, если переменная была к этому времени определена, ее значение используется обычным образом.

2.5.3. Экспорт переменных

В ОС UNIX существует понятие процесса. Процесс возникает тогда, когда запускается на выполнение какая-либо команда (расчет). Например, при наборе на клавиатуре "p" порождается процесс расчета p. В свою очередь p может породить другие процессы. Допустим, что p вызывает расчеты p1 и p2, которые последовательно порождают соответствующие процессы.

У каждого процесса есть своя среда - множество доступных ему переменных. Например, до запуска расчета p уже существовала среда, в которой уже были определены некоторые переменные. Запуск p порождает новую среду; уже в ней будут порождены расчеты p1 и p2.

Переменные локальны в рамках процесса, в котором они объявлены, т.е. где им присвоены значения (описание переменных отсутствует - они все одного типа). Для того, чтобы они были доступны и другим порождаемым процессам, надо передать их явным образом. Для этого используется встроенная команда export.

Пример. Пусть расчет (командный файл) p, имеющий вид:

 # расчет p
echo Расчет p
varX=0 varY=1
echo varX=$varX varY=$varY
export varY
p1 # вызов расчета p1
p2 # вызов расчета p2
echo Снова расчет p: varX=$varX varY=$varY

вызывает командные файлы "p1" и "p2", имеющие вид:

 # расчет p1
echo Расчет p1
echo varX=$varX varY=$varY
varX=a varY=b
echo varX=$varX varY=$varY
export varX

 # расчет p2
echo Расчет p2
echo varX=$varX varY=$varY varX=A varY=B
echo varX=$varX varY=$varY
export varY

На экран будет выдана следующая информация:

Расчет p
varX=0 varY=1
Расчет p1
varX= varY=1
varX=a varY=b
Расчет p2
varX= varY=1
varX=A varY=B
Снова расчет p:
varX=0 varY=1

Из примера видно, что значения переменных экспортируются только в вызываемые расчеты (и не передаются "вверх" и "вбок"). Экспортировать переменные можно и командой set с флагом -a.

На всякий случай заметим, что на передачу значений переменных никакого влияния не оказывает "физическое" взаимное расположение файлов расчетов в файловой системе.

2.5.4. Параметры

В командный файл могут быть переданы параметры. В Shell используются позиционные параметры (т.е. существенна очередность их следования). В командном файле соответствующие параметрам переменные (аналогично Shell-переменным) начинаются с символа "$", а далее следует одна из цифр от 0 до 9. Пусть расчет examp-1 вызывается с параметрами "cock" и "tail". Эти параметры попадают в новую среду под стандартными именами "1" и "2". В (стандартной) переменной с именем "0" будет храниться имя вызванного расчета.

При обращении к параметрам перед цифрой ставится символ доллара "$" (как и при обращении к переменным):

$0 - соответствует имени данного командного файла;

$1 - первый по порядку параметр;

$2 - второй параметр и т.д.

Пусть командный файл с именем "examp-1" имеет вид

echo Это расчет $0:

sort $2 >> $1

cat $1

а файлы "cock" и "tail" содержат соответственно

cock:

Это отсортированный файл:

и

tail:

1

2

3

Тогда после вызова команды

examp-1 cock tail

на экране будет

Это расчет examp-1:
Это отсортированный файл:
1
2
3

Поскольку число переменных, в которые могут передаваться параметры, ограничено (не более 9; "0", как уже отмечалось, имеет особый смысл), то для передачи большего числа параметров используется специальная команда shift. Рассмотрим ее действие на примере. Пусть командный файл many вызывается с 13 параметрами

many 10 20 30 40 50 60 70 80 90 100 110 120 130

и имеет вид

```
###
# many: Передача большого числа параметров.
echo "$0: Много параметров"
echo " Общее число параметров = $#
Исходное состояние: $1 $5 $9 "
shift
echo "1 сдвиг: первый=$1 пятый=$5 девятый=$9"
shift 2
echo "1 + 2 = 3 сдвига: первый=$1 пятый=$5 девятый=$9"
perem=`expr $1 + $2 + $3`
echo $perem
```

В результате первого применения команды shift второй параметр расчета вызывается как $1, третий параметр вызывается как $2, ... десятый параметр, который был исходно недоступен, вызывается как $9. Но стал недоступным первый параметр! После выполнения этого расчета на экране будет:

many: Много параметров Общее число параметров = 13
Исходное состояние: 10 50 90
1 сдвиг: первый=20 пятый=60 девятый=100
1 + 2 = 3 сдвига: первый=40 пятый=80 девятый=120
150

Своеобразный подход к параметрам дает команда set. Например, фрагмент расчета

```
set a b c
echo первый=$1 второй=$2 третий=$3
```

выдаст на экран

первый=a второй=b третий=c

т.е. команда "set" устанавливает значения параметров. Это бывает очень удобно. Например, команда "date" выдает на экран текущую дату, скажем, "Mon May 01 12:15:10 2000", состоящую из пяти слов, тогда

set `date`
echo $1 $3 $5

выдаст на экран

Mon 01 2000

Команда "set" позволяет также осуществлять контроль выполнения программы, например:

set -v - на терминал выводятся строки, читаемые Shell;

set +v - отменяет предыдущий режим;

set -x - на терминал выводятся команды перед выполнением;

set +x - отменяет предыдущий режим.

Команда "set" без параметров выводит на терминал состояние программной среды.

2.5.5. Подстановки Shell-интерпретатора

Перед началом непосредственной интерпретации и выполнением команд, содержащихся в командных файлах, Shell выполняет различные виды подстановок:

1. ПОДСТАНОВКА РЕЗУЛЬТАТОВ. Выполняются все команды, заключенные в обратные кавычки, и на их место подставляется результат.
2. ПОДСТАНОВКА ЗНАЧЕНИЙ ПАРАМЕТРОВ И ПЕРЕМЕННЫХ. То есть слова, начинающиеся на "$", заменяются соответствующими значениями переменных и параметров.
3. ИНТЕРПРЕТАЦИЯ ПРОБЕЛОВ. Заэкранированные пробелы игнорируются.
4. ГЕНЕРАЦИЯ ИМЕН ФАЙЛОВ. Проверяются слова на наличие в них спецсимволов ("*", "?","[]") и выполняются соответствующие генерации.

2.6. Программная среда

Каждый процесс имеет среду, в которой он выполняется. Shell использует ряд переменных этой среды. Если вы наберете команду "set" без параметров, то на экран будет выдана информация о ряде стандартных переменных, созданных при входе в систему (и передаваемых далее всем вашим новым процессам "по наследству"), а также переменных, созданных и экспортируемых вашими процессами. Конкретный вид и содержание выдаваемой информации в немалой степени зависит от того, какая версия UNIX используется и как инсталлирована система.

Вот лишь часть того, что может выдать команда "set":

```
HOME=/home/sae
PATH=/usr/local/bin:/usr/bin:/bin:.:/usr/bin/X11:
IFS=

LOGNAME=sae
MAIL=/var/spool/mail/sae
PWD=/home/sae/STUDY/SHELL
PS1=${PWD}:" "
PS2=>
SHELL=/bin/bash
TERM=Linux
TERMCAP=console|con80x25|dumb|Linux:li#25:co#80::
UID=501
perem=stroka
x=5
```

Прокомментируем эти присваивания значений переменным.

HOME=/home/sae - это имя домашнего каталога.

PATH=/usr/local/bin:/usr/bin:/bin:.:/usr/bin/X11 – эта переменная задает последовательность файлов (тропу), которые просматривает Shell в поисках команды. Имена файлов разделяются здесь двоеточиями. Последовательность просмотра соответствует очередности следования имен в тропе. Но первоначально поиск происходит среди так называемых встроенных команд. В число встроенных команд входят наиболее часто используемые команды, например echo, cd, pwd, date. После этого система просматривает каталог /bin, в котором могут находиться команды sh, cp, mv, ls и т.п., затем каталог /usr/bin с командами cat, cc, expr, nroff, man и многими другими. Далее поиск происходит в текущем каталоге ("." или другое обозначе-

ние - "пусто", т.е.""), где скорее всего находятся написанные вами команды (расчеты).

После набора командной строки и нажатия клавиши Enter (после выполнения необходимых подстановок) интерпретатор распознает имя, соответствующее команде, и осуществляет ее поиск в каталогах, перечисленных в тропе. Если команда размещена вне этих каталогов - она не будет найдена. Если присутствует несколько команд с одинаковым именем, то вызвана будет та, которая расположена в каталоге, просматриваемом первым. Тропу, как и прочие переменные, можно легко менять, добавляя, переставляя или исключая каталоги.

IFS= - (внутренний разделитель полей) перечисляет символы, которые служат для разделения слов (полей). Таковыми являются "пробел", "табуляция" и "перевод строки", поэтому здесь слева от присваивания ничего не видно и занято две строки.

LOGNAME=sae - имя входа ("имя" пользователя).

MAIL=/var/spool/mail/sae - имя файла, в который поступает электронная почта.

PWD=/home/sae/STUDY/SHELL - имя текущего каталога.

PS1=${PWD}:" " - вид промтера. В данном случае в промтере будет выдаваться имя текущего каталога, двоеточие и пробел. То есть здесь будет "/home/sae/STUDY/SHELL: ".

PS2=> - этот промтер (здесь ">") используется как приглашение к продолжению ввода (в очередной строке) незаконченной команды.

SHELL=/bin/bash - эта переменная указывает оболочку, которую использует пользователь.

TERM=Linux - указание типа терминала.

TERMCAP=console|con80x25|dumb|Linux:li#25:co#80:: - (TERMinal CAPacity) это очень сильно обрезанная строка задания параметров терминала.

UID=501 - идентификатор пользователя.

perem=stroka x=5 - переменные, которые ввел пользователь.

Исходная среда устанавливается автоматически при входе в систему с использованием файлов типа "/etc/rc" и "/etc/.profile".

Один из способов просто изменить среду (например, тропу поиска команд, вид промтера, вид оболочки, цвет экрана и т.п.) – разместить эту информацию в своем домашнем каталоге в специализированном файле ".profile" (${HOME}/.profile), присвоив нужные значения переменным среды. То есть вызвать это файл в редактор и написать, что пожелаете. Тогда при каждом вашем входе в систему этот файл будет автоматически выполняться и устанавливать новую среду. Этот файл должен *обязательно* раз-

мещаться в вашем *домашнем* каталоге. Если вы внесли изменения в ".profile", то для переноса этих изменений в среду необходимо выполнить этот файл. Для этого можно выйти и заново войти в систему, а можно воспользоваться командой **.** (точка) без выхода из системы, т.е.

. .profile

Следует иметь в виду, что имена файлов, начинающиеся с точки, вообще имеют особый статус. Так, они не выдаются на экран простой командой ls - необходимо вызывать эту команду с флагом "-a". Кстати, и не уничтожаются огульно командой rm *. Дописать новый каталог "my" в тропу команд можно, записав в .profile, например,

PATH=${PATH}:/home/sae/my

или

PATH=${PATH}:${HOME}/my

Как правило, устанавливаемые переменные среды следует экспортировать. Например

export TERM PATH REDKEYS MAIL

Кроме определения переменных в .profile можно выполнить команды, например команда stty -lcase установит терминал в режим "большие и маленькие буквы". Сам интерпретатор Shell автоматически присваивает значения следующим переменным (параметрам):

? - значение, возвращенное последней командой;

$ - номер процесса;

! - номер фонового процесса;

- число позиционных параметров, передаваемых в Shell;

* - перечень параметров как одна строка;

@ - перечень параметров как совокупность слов;

- - флаги, передаваемые в Shell.

При обращении к этим переменным (т.е при использовании их в командном файле - Shell-программе) следует впереди ставить "$". Пример. Вызов расчета

specific par1 par2 par3

имеющего вид

specific: Специальные параметры (переменные)
echo $0 - имя расчета
echo $? - код завершения
echo $$ - идентификатор последнего процесса
echo $! - идентификатор последнего фонового процесса
echo echo $* - значения параметров как строки
echo $@ - значения параметров как слова
echo set –au

echo $- - режимы работы интерпретатора
выдаст на экран

specific - имя расчета
0 - код завершения
499 - идентификатор последнего процесса
98 - идентификатор последнего фонового процесса
par1 par2 par3 - значения параметров как строки
par1 par2 par3 - значения параметров как слова
au - режимы работы интерпретатора

Код "0" соответствует нормальному завершению процесса. Важную роль при создании уникальных файлов играет специальная переменная "$$", значение которой соответствует номеру процесса, выполняющего данный расчет. Каждый новый расчет, выполняемый компьютером, инициирует один или несколько процессов, автоматически получающих номера по порядку. Поэтому, используя номер процесса в качестве имени файла, можно быть уверенным, что каждый новый файл будет иметь новое имя (не запишется на место уже существующего). Достоинство является и главным недостатком такого способа именования файлов. Неизвестно, какие имена будут присвоены файлам. И, если в рамках данного процесса можно найти файл "не глядя", т.е. обратившись к нему, используя $$, то потом такие файлы можно легко потерять. Это создает дополнительные проблемы при отладке программ.

"echo" без параметров выводит пустую строку.

Различие $* и $@ состоит в том, что первая переменная может быть представлена как "par1 par2 par3", а вторая как "par1" "par2" "par3". Пример, иллюстрирующий различие "$*" и "$@", будет рассмотрен в связи с оператором for. Для иллюстрации мы установили командой set режимы интерпретатора ("a" - все последующие переменные экспортируются; "u" - отсутствие параметра считать ошибкой), что и отразилось в специальной переменной "$-".

2.7. Программные структуры

Как во всяком языке программирования в тексте на языке Shell могут быть комментарии. Для этого используется символ "#". Все, что находится в строке (в командном файле) правее этого символа, воспринимается интерпретатором как комментарий. Например,

Это комментарий.

И это.

И это то же.

Как во всяком процедурном языке программирования в языке Shell есть операторы. Ряд операторов позволяет управлять последовательностью выполнения команд. В таких операторах часто необходима проверка условия, которая и определяет направление продолжения вычислений.

2.7.1. Команда test ([ ])

Команда test проверяет выполнение некоторого условия. С использованием этой (встроенной) команды формируются операторы выбора и цикла языка Shell. Два возможных формата команды:

test условие

или

[условие]

На самом деле Shell будет распознавать эту команду по открывающей скобке "[" как слову, соответствующему команде test. Уже этого достаточно, чтобы предупредить о распространенной ошибке начинающих: между скобками и содержащимся в них условием обязательно должны быть пробелы. Пробелы должны быть и между значениями и символом сравнения или операции (как, кстати, и в команде expr). Не путать с противоположным требованием для присваивания значений переменным.

В Shell используются условия различных типов.

УСЛОВИЯ ПРОВЕРКИ ФАЙЛОВ:

-f file - файл "file" является обычным файлом;
-d file - файл "file" - каталог;
-c file - файл "file" - специальный файл;
-r file - имеется разрешение на чтение файла "file";
-w file - имеется разрешение на запись в файл "file";
-s file - файл "file" не пустой.

Примеры. Вводя с клавиатуры командные строки, в первом случае получим подтверждение (код завершения "0"), а во втором - опровержение (код завершения "1"). specific - имя существующего файла.

[-f specific] ; echo $?

0

[-d specific] ; echo $?

1

УСЛОВИЯ ПРОВЕРКИ СТРОК

str1 = str2 - строки "str1" и "str2" совпадают;

str1 != str2 - строки "str1" и "str2" не совпадают;
-n str1 - строка "str1" существует (непустая);
-z str1 строка "str1" не существует (пустая).

Примеры.

x="who is who"; export x; ["who is who" = "$x"]; echo $?

0

x=abc ; export x ; [abc = "$x"] ; echo $?

0

x=abc ; export x ; [-n "$x"] ; echo $? 0 x="" ; export x ; [-n "$x"] ; echo $?

1

ВАЖНОЕ ЗАМЕЧАНИЕ. Команда test дает значение "истина" (т.е. код завершения "0") и просто если в скобках стоит непустое слово.

[privet] ; echo $?

0

[ ] ; echo $?

1

Кроме того, существуют два стандартных значения условия, которые могут использоваться вместо условия (для этого не нужны скобки).

true ; echo $?

0

false ; echo $?

1

УСЛОВИЯ СРАВНЕНИЯ ЦЕЛЫХ ЧИСЕЛ

x -eq y - "x" равно "y",
x -ne y - "x" не равно "y",
x -gt y - "x" больше "y",
x -ge y - "x" больше или равно "y",
x -lt y - "x" меньше "y",
x -le y - "x" меньше или равно "y".

То есть в данном случае команда test воспринимает строки и символы как целые числа. Поэтому во всех остальных случаях "нулевому" значению соответствует пустая строка. В данном же случае, если надо обнулить переменную, скажем, "x", то это достигается присваиванием "x=0".

Примеры.

x=abc ; export x ; [abc -eq "$x"] ; echo $?

"[": integer expression expected before –eq

x=321 ; export x ; [321 -eq "$x"] ; echo $?

0

x=3.21 ; export x ; [3.21 -eq "$x"] ; echo $?
"[": integer expression expected before –eq
x=321 ; export x ; [123 -lt "$x"] ; echo $?
0

СЛОЖНЫЕ УСЛОВИЯ

Реализуются с помощью типовых логических операций:

! - (not) инвертирует значение кода завершения;
-o - (or) соответствует логическому "ИЛИ";
-a - (and) соответствует логическому "И".

ПРЕДУПРЕЖДЕНИЕ. Не забывайте о пробелах. Примеры.

[! privet] ; echo $?
1
x=privet; export x; ["$x" -a -f specific] ; echo $?
0
x="";export x; ["$x" -a -f specific] ; echo $?
1
x="";export x; ["$x" -a -f specific -o privet] ; echo $?
0
x="";export x; ["$x" -a -f specific -o ! privet] ; echo $?
1

2.7.2. Условный оператор "if"

В общем случае оператор "if" имеет структуру

if условие
then список
[elif условие
then список]
[else список]
fi

Здесь "elif" (сокращенный вариант от "else if") может быть использован наряду с полным, т.е. допускается вложение произвольного числа операторов "if" (как и других операторов). Разумеется, "список" в каждом случае должен быть осмысленный и допустимый в данном контексте. Конструкции [elif условие then список] и [else список] не являются обязательными (в данном случае для указания на необязательность конструкций использованы квадратные скобки - не путать с квадратными скобками команды test!). Самая усеченная структура этого оператора

if условие
then список
fi

Если выполнено условие, то выполняется "список", иначе он пропускается. Обратите внимание, что структура обязательно завершается служебным словом fi. Число fi, естественно, всегда должно соответствовать числу if. Примеры. Пусть написан расчет if-1:

if [$1 -gt $2]
then pwd
else echo $0 : Hello!
fi

Тогда вызов расчета if-1 12 11 даст

/home/sae/STUDY/SHELL

a if-1 12 13 даст

if-1 : Hello!

Возможно использовать в условии то свойство Shell, что команды могут выдавать различный код завершения. Пусть расчет if-2 будет следующим:

if a=`expr "$1" : "$2"`
then echo then a=$a code=$?
else echo else a=$a code=$?
fi

Тогда вызов if-2 by by даст

then a=2 code=0

a if-2 by be даст

else a=0 code=1

Еще пример на вложенность

if-3: Оценка достижений echo -n " А какую оценку получил на экзамене?: "
read z
if [$z = 5]
then echo Молодец !

```
   elif [ $z = 4 ]
        then echo Все равно молодец !
        elif [ $z = 3 ]
             then echo Все равно !
             elif [ $z = 2 ]
                  then echo Все !
                  else echo !
fi
```

Обратите внимание на то, что желательно использовать сдвиги при записи программ, чтобы лучше выделить вложенность структур.

2.7.3. Оператор выбора "case"

Оператор выбора "case" имеет структуру:

```
case строка in
     шаблон) список команд;;
     шаблон) список команд;;
...
esac
```

Здесь "case", "in" и "esac" - служебные слова. "Строка" (это может быть и один символ) сравнивается с "шаблоном". Затем выполняется "список команд" выбранной строки. Непривычным будет служебное слово "esac", но оно необходимо для завершения структуры. Пример.

```
### # case-1: Структура "case".
# Уже рассматривавшийся в связи со
# структурой "if" пример проще и
# нагляднее можно реализовать с
# помощью структуры "case".
echo -n " А какую оценку получил на экзамене?: "
read z
case $z in
     5) echo Молодец ! ;;
     4) echo Все равно молодец ! ;;
     3) echo Все равно ! ;;
     2) echo Все ! ;;
     *) echo ! ;;
esac
```

Непривычно выглядят ";;" в конце строк выбора, но написать здесь ";" было бы ошибкой. Для каждой альтернативы может быть выполнено несколько команд. Если эти команды будут записаны в одну строку, то сим-

вол ";" будет использоваться как разделитель команд. Обычно последняя строка выбора имеет шаблон "*", что в структуре case означает "любое значение". Эта строка выбирается, если не произошло совпадение значения переменной ни с одним из ранее записанных шаблонов, ограниченных скобкой ")". Значения просматриваются в порядке записи.

```
### # case-2: Справочник.
# Для различных фирм по имени выдается
# название холдинга, в который она входит
case $1 in
    ONE|TWO|THREE) echo Холдинг: ZERO ;;
    MMM|WWW) echo Холдинг: Not-Net ;;
    Hi|Hello|Howdoing) echo Холдинг: Привет! ;;
    *) echo Нет такой фирмы ;;
esac
```

При вызове "case-2 Hello" на экран будет выведено:

Холдинг: Привет!

А при вызове "case-2 HELLO" на экран будет выведено:

Нет такой фирмы

Коль скоро слово "case" переводится как "выбор", то это как бы намек на то, что можно эту структуру использовать для реализации простейших меню.

```
### # case-3: Реализация меню с помощью команды  case
echo "Назовите файл, а затем (через пробел) наберите цифру, соответствующую требуемой обработке:
1 – отсортировать
2 - выдать на экран
3 - определить число строк "
read x y  # x - имя файла, y - что сделать
case $y in
    1)  sort < $x ;;
    2)  cat < $x ;;
    3)  wc -l < $x ;;
    *)  echo " Мы не знаем такой команды ! " ;;
esac
```

Разумеется, желания могут быть более сложные и на месте отдельных команд могут быть последовательности команд или вызовы более сложных

расчетов. Напишем команду case-4, которая добавляет информацию к файлу, указанному первым параметром (если параметр один), со стандартного входа либо (если 2 параметра) из файла, указанного в качестве первого параметра:

```
### # case-4: Добавление в файл.
# Использование стандартной переменной.
# "$#" - число параметров при вводе расчета
# ">>" - перенаправление с добавлением в файл
case $# in
     1)  cat >> $1 ;;
     2)  cat >> $2 < $1 ;;
     *)  echo "Формат: case-4 [откуда] куда" ;;
esac
```

"$1" (при "$#=1") - это имя файла, в который происходит добавление со стандартного входа.

"$1" и "$2" (при $#=2) - это имена файлов, из которого ("$1") и в который ("$2") добавлять. Во всех других случаях (*)) выдается сообщение о том, каким должен быть правильный формат команды.

2.7.4. Оператор цикла с перечислением "for"

Оператор цикла "for" имеет структуру:

```
for имя [in список значений]
    do список команд
    done
```

где "for" - служебное слово, определяющее тип цикла, "do" и "done" - служебные слова, выделяющие тело цикла. Не забывайте про "done"! Фрагмент "in список значений" может отсутствовать. Пусть команда lsort представлена командным файлом

```
for i in f1 f2 f3
    do proc-sort $i
    done
```

В этом примере имя "i" играет роль параметра цикла. Это имя можно рассматривать как Shell-переменную, которой последовательно присваиваются перечисленные значения (i=f1, i=f2, i=f3), и выполняется в цикле команда "procsort". Часто используется форма "for i in *", означающая "для всех файлов текущего каталога". Пусть proc-sort, в свою очередь, представляется командным файлом

```
cat $1 | sort | tee /dev/lp > ${1}_sorted
```

т.е. последовательно сортируются указанные файлы, результаты сортировки выводятся на печать ("/dev/lp") и направляются в файлы f1_sorted f2_sorted и f3_sorted. Можно сделать более универсальной команду lsort, если не фиксировать перечень файлов в команде, а передавать произвольное их число параметрами. Тогда головная программа будет следующей:

```
for i
    do proc-sort $i
    done
```

Здесь отсутствие после "i" служебного слова "in" с перечислением имен говорит о том, что список поступает через параметры команды. Результат предыдущего примера можно получить, набрав lsort f1 f2 f3.

Усложним ранее рассматривавшуюся задачу (под именем "case-2") определения холдинга фирмы. Теперь можно при вызове указывать произвольное количество фирм. При отсутствии в структуре оператора for фрагмента "in список значений" значения берутся из параметров вызывающей команды.

```
### # holding: Справочник.
# Для различных фирм по имени выдается
# название холдинга, в который она входит
for i
    do case $i in
            ONE|TWO|THREE) echo Холдинг: ZERO ;;
            MMM|WWW) echo Холдинг: Not-Net ;;
            Hi|Hello|Howdoing) echo Холдинг: Привет! ;;
            *) echo Нет такой фирмы ;;
       esac
    done
```

При вызове "holding Hello HELLO ONE" на экране будет:

```
Холдинг: Привет!
Нет такой фирмы
Холдинг: Not-Net
```

Еще пример.

```
### # subdir: Выдает имена всех подкаталогов
# каталога с именем $dir
for i in $dir/*
    do if [ -d $i ]
          then echo $i
       fi
```

```
    done
```

Следующий расчет иллюстрирует полезный, хотя и с долей трюкачества, способ повторения одних и тех же действий несколько раз. Переменная "i" принимает здесь пять значений: 1, 2, 3, 4, 5, но внутри цикла эта переменная отсутствует, и поэтому ее значение никакой роли не играет и ничего не меняет.

```
### # print-5: Организации пятикратного выполнения команды
for i in 1 2 3 4 5
    do cat file-22 > /dev/lp
    done
```

Расчет "print-n" иллюстрирует еще одну полезную возможность в использовании цикла "for". Здесь, после "for i ...", отсутствуют "in ..." и перечень имен, т.е. перечнем имен для "i" становится перечень параметров, а следовательно количество печатаемых экземпляров можно менять.

```
### # print-n: Задание числа копий
# через параметры
for i
    do cat file-22 > /dev/lp
    done
```

Смысл не изменится, если первую строку расчета записать как for i in $*, поскольку значение "$*" - есть список значений параметров. Отметим различие в специальных переменных "$*" и "$@", представляющих перечень параметров. Первая представляет параметры, как строку, а вторая - как совокупность слов. Пусть командный файл cmp имеет вид:

```
for i in "$*"
    do echo $i
    done
echo
for i in "$@"
    do echo $i
    done
```

При вызове cmp aa bb cc на экран будет выведено

```
aa bb cc

aa
bb
cc
```

2.7.5. Оператор цикла с истинным условием "while"

Структура "while", также обеспечивающая выполнение расчетов, предпочтительнее тогда, когда не известен заранее точный список значений параметров или этот список должен быть получен в результате вычислений в цикле. Оператор цикла "while" имеет структуру:

```
while условие
        do список команд
        done
```

где "while" - служебное слово определяющее тип цикла с истинным условием. Список команд в теле цикла (между "do" и "done") повторяется до тех пор, пока сохраняется истинность условия (т.е. код завершения последней команды в теле цикла равен 0) или цикл не будет прерван изнутри специальными командами (break, continue или exit). При первом входе в цикл условие должно выполняться.

```
### # print-50: Структура "while"
# Расчет позволяет напечатать 50
# экземпляров файла "file-22"
n=0
while [ $n -lt 50 ] # пока n < 50
        do n=`expr $n + 1` cat file-22 > /dev/lp
        done
```

Обратим внимание на то, что переменной "n" вначале присваивается значение 0, а не пустая строка, так как команда expr работает с Shell-переменными как с целыми числами, а не как со строками.

n=`expr $n + 1`

т.е. при каждом выполнении значение "n" увеличивается на 1. Можно реализовать то же самое и сложнее. Расчет pr-br приведен для иллюстрации бесконечного цикла и команды break для прекращения цикла.

```
### # pr-br: Структура "while" с "break"
# Расчет позволяет напечатать 50
# экземпляров файла "file-22"
n=0
while true
        do if [ $n -lt 50 ] # если n < 50
              then n=`expr $n + 1`
              else break
           fi
           cat file-22 > /dev/lp
        done
```

Команда "break [n]" позволяет выходить из цикла. Если "n" отсутствует, то это эквивалентно "break 1"; "n" указывает число вложенных циклов, из которых надо выйти, например "break 3" - выход из трех вложенных циклов. В отличие от команды "break" команда "continue [n]" лишь прекращает выполнение текущего цикла и возвращает на *начало* цикла. Она также может быть с параметром. Например, "continue 2" означает выход на начало второго (если считать из глубины) вложенного цикла.

Команда "exit [n] " позволяет выйти вообще из процедуры с кодом возврата "0" или "n" (если параметр "n" указан). Эта команда может использоваться не только в циклах. Даже в линейной последовательности команд она может быть полезна при отладке, чтобы прекратить выполнение (текущего) расчета в заданной точке.

2.7.6. Оператор цикла с ложным условием "until"

Оператор цикла "until" имеет структуру:

```
until условие
    do список команд
    done
```

где "until" - служебное слово, определяющее тип цикла с ложным условием. Список команд в теле цикла (между "do" и "done") повторяется до тех пор, пока сохраняется ложность условия или цикл не будет прерван изнутри специальными командами (break, continue или exit). При первом входе в цикл условие не должно выполняться. Отличие от оператора while состоит в том, что условие цикла проверяется на ложность (на ненулевой код завершения последней команды тела цикла) *после* каждого (в том числе и первого!) выполнения команд тела цикла. Программистов, знакомых с оператором "until" в других языках, может вначале сбивать такая семантика этого оператора. Примеры.

```
until false
    do read x
       if [ $x = 5 ]
          then echo enough ; break
          else echo some more
       fi
    done
```

Здесь программа с бесконечным циклом ждет ввода слов (повторяя на экране фразу "some more"), пока не будет введено "5". После этого выдается enough и команда break прекращает выполнение цикла.

Другой пример ("Ожидание полдня") иллюстрирует возможность использовать в условии вычисления.

```
until date | grep 12:00:
      do sleep 30
      done
```

Здесь каждые 30 секунд выполняется командная строка условия. Команда date выдает текущую дату и время. Команда grep получает эту информацию через конвейер и пытается совместить заданный шаблон "12:00:" с временем, выдаваемым командой date. При несовпадении grep выдает код возврата "1", что соответствует значению "ложь", и цикл "выполняет ожидание" в течение 30 секунд, после чего повторяется выполнение условия. В полдень (возможно с несколькими секундами) произойдет сравнение, условие станет истинным, "grep" выдаст на экран соответствующую строку и работа цикла закончится.

2.7.7. Пустой оператор

Пустой оператор имеет формат

```
;
```

Ничего не делает. Возвращает значение "0". Используется, например, в конструкции "while ;" или ставится в начале командного файла, чтобы гарантировать, что файл не будет принят за выполняемый файл для "csh".

2.7.8. Функции в Shell

Функция позволяет подготовить список команд Shell для последующего выполнения. Описание функции меет вид:

```
имя()
{
 список команд
}
```

После чего обращение к функции происходит по имени. При выполнении функции не создается нового процесса. Она выполняется в среде соответствующего процесса. Аргументы функции становятся ее позиционными параметрами; имя функции - ее нулевой параметр. Прервать выполнение функции можно оператором "return [n]", где (необязательное) n - код возврата. Пример. Вызов на выполнение файла fun

```
echo $$
fn() # описание функции
{
```

```
 echo xx=$xx
 echo $#
 echo $0: $$ $1 $2
 xx=yy ; echo xx=$xx
 return 5
}
xx=xx ; echo xx=$xx
fn a b # вызов функции "fn" с параметрами
echo $?
echo xx=$xx
```

содержащего описание и вызов функции fn, выдаст на экран:

```
749
xx=xx
xx=xx
2
fun: 749 a b
xx=yy
5
xx=yy
```

2.7.9. Примеры простых программ на Shell.

Пример 1

```
#!/bin/sh
cat $1 | (
TTT=0
NNN=0
while read a0 a1 a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a13 a14 a15
do
TTT=`echo "$TTT + $a14" | bc -q`
NNN=`expr $NNN + 1`
done
```

```
        echo "----------------------------------------"
        echo "Total:       $NNN Clients      $TTT Profit"
        echo "$1:       $NNN Clients      $TTT Profit" >>
/usr/account/RESULTS/SUM.REG
        ) >> $1
        exit 0
```

Пример 2

```
        #!/bin/sh
         finger @as5300 | grep $1 >/usr/account/tmp/Finger.tmp
         if [ $? = 0 ]; then
          echo "as5300"
          cat /usr/account/tmp/Finger.tmp
         fi
         finger @tts-11-1 | grep $1 >/usr/account/tmp/Finger.tmp
         if [ $? = 0 ]; then
          echo "tts-11-1"
          cat /usr/account/tmp/Finger.tmp
         fi
         finger @asu-11-1 | grep $1 >/usr/account/tmp/Finger.tmp
         if [ $? = 0 ]; then
          echo "asu-11-1"
          cat /usr/account/tmp/Finger.tmp
         fi
         finger @asu-11-2 | grep $1 >/usr/account/tmp/Finger.tmp
         if [ $? = 0 ]; then
          echo "asu-11-2"
          cat /usr/account/tmp/Finger.tmp
         fi
```

```
#  finger @asu-11-5 | grep $1 >/usr/account/tmp/Finger.tmp
# if [ $? = 0 ]; then
#  echo "asu-11-5"
#  cat /usr/account/tmp/Finger.tmp
# fi
# finger @asu-11-6 | grep $1 >/usr/account/tmp/Finger.tmp
# if [ $? = 0 ]; then
#  echo "asu-11-6"
#  cat /usr/account/tmp/Finger.tmp
# fi
exit 0
#
```

Пример 3

```
#!/bin/sh

DIR=/usr/account
CC=0
LC=1
while test $CC = 0
do
 sed -n $LC,1p $1 >/usr/account/adm/TTT.tmp
 read a b c d </usr/account/adm/TTT.tmp
if [ x$a = x";" ]; then exit 0; fi
if [ -d /usr/account/users/$a ]; then
 case $a in
    \#*) continue;;
    \;*) CC=1
            break;;
```

```
            *) echo "Информация по текущему сальдо $b"
>/usr/account/templates/warning.tpls
                echo "" >>/usr/account/templates/warning.tpls
                cat
/usr/account/templates/warning.tpl>>/usr/account/templates/warning.tpls
                echo -n "$d " >>/usr/account/templates/warning.tpls
            cat
/usr/account/templates/warning.tpl0>>/usr/account/templates/warning.tpls
                echo -n """>>/usr/account/templates/warning.tpls
            cat
/usr/account/templates/warning.tpl1>>/usr/account/templates/warning.tpls
            mv /usr/account/templates/warning.tpls /usr/account/users/$a/warn.txt
            /usr/account/bin/sendresult $a /usr/account/users/$a/warn.txt
                echo "$a - warning sent `date`"
>>/usr/account/adm/spool/warning.log
                echo "$a - warning sent";;
        esac
       else
       echo "$a - Directory not found `date`"
>>/usr/account/adm/spool/warning.log
       echo "Directory $a - not found"
       fi
       LC=`expr $LC + 1`
       sleep 2
       done
       exit 0
```

2.7.10. Пример сложной программы на Shell.

```
# Add IP & POP3 user to accounting system
if [ -f /usr/account/locks/Add.Ip.user ]
 then
 echo "User `cat /usr/account/locks/Add.Ip.user` already works with regis-
tration ! Wait...and retry."
 exit 1
fi
whoami >/usr/account/locks/Add.Ip.user
read uu </usr/account/locks/Add.Ip.user
if [ x$uu != x"acnt" ]
 then
 if [ x$uu != x"registr" ]
  then
  echo "Error: this program may start ONLY under login "acnt" !"
  rm /usr/account/locks/Add.Ip.user
  exit 1
 fi
fi
dialog --title "Create IP & POP3 Account" --clear \
--inputbox "Enter user hostname (without elcom.ru !!!):" 8 60 2>/tmp/a.$$
a=`cat /tmp/a.$$`
rm -f /tmp/a.$$
if [ x$a = x ]; then
  dialog --title "Create IP & POP3 Account" --clear --msgbox \
  "Void input is not allowed!" 5, 40
  clear
```

```
            rm /usr/account/locks/Add.Ip.user
          exit 1
        fi
        /usr/account/adm/check.dns $a
        if [ $? = 0 ]
                    then
                  dialog --title "Create IP & POP3 Account" --clear \
                  --msgbox "Error: domain $a.elcom.ru already exists !!!\nTry again,
my friend !" 6 60
                  clear
                     rm /usr/account/locks/Add.Ip.user
                     exit 1
        fi
        UserList=`cat /usr/account/USERLIST`
        for i in $UserList; do
         case $i in
           \#*) continue;;
           \;*)break;;
             *) if [ $a = $i ]
                    then
                  dialog --title "Create IP & POP3 Account" --clear \
                  --msgbox "Error: domain $a.elcom.ru already exists !!!\nTry again,
my friend !" 6 60
                  clear
                     rm /usr/account/locks/Add.Ip.user
                     exit 1
                   fi;;
        esac
        done
        UserList=`cat /usr/account/users/LOCKED/LLIST`
```

```
for i in $UserList; do
 case $i in
   \#*) continue;;
   \;*)break;;
    *) if [ $a = $i ]
         then
       dialog --title "Create IP & POP3 Account" --clear \
       --msgbox "Error: domain $a.elcom.ru already exists !!!\nTry again,
my friend !" 6 60
       clear
         rm /usr/account/locks/Add.Ip.user
         exit 1
       fi;;
esac
done
UserList=`cat /usr/account/users/SERVICE/SLIST`
for i in $UserList; do
 case $i in
   \#*) continue;;
   \;*)break;;
    *) if [ $a = $i ]
         then
       dialog --title "Create IP & POP3 Account" --clear \
       --msgbox "Error: domain $a.elcom.ru already exists !!!\nTry again,
my friend !" 6 60
       clear
         rm /usr/account/locks/Add.Ip.user
         exit 1
       fi;;
esac
```

```
done
if [ -d /usr/account/users/$a ]; then
 dialog --title "Create IP & POP3 Account" --clear \
 --msgbox "Error: Directory /usr/account/users/$a already exists !!!\nTry
again, my friend !" 6 75
 clear
                rm /usr/account/locks/Add.Ip.user
 exit 1
fi
date | /usr/account/bin/mkuserdate
DD=`cat /usr/account/spool/Date.user`
dialog --title "Create IP & POP3 Account" --clear \
--inputbox "Enter organization name:" 8 60 2>/tmp/on.$$
on=`cat /tmp/on.$$`
rm /tmp/on.$$
if [ x$on = x ]; then
  dialog --title "Create IP & POP3 Account" --clear --msgbox \
  "Void input is not allowed!" 5, 40
  clear
                rm /usr/account/locks/Add.Ip.user
  exit 1
fi
dialog --title "Create IP & POP3 Account" --clear \
--menu "Select Payment manner:" 10 40 5 "o" "Internet Online" "f" \
"Full Time Unlimited" "n" "Night Time Unlimited" "c" "Full Unlimited"
2>/tmp/e.$$
e=`cat /tmp/e.$$`
rm -f /tmp/e.$$
 case $e in
    o*) ii="Internet online";DT="3.33";break;;
```

```
    f*) ii="Full Time unlimited";DT="26.67";break;;
    n*) ii="Night Time unlimited";DT="11.67";break;;
    c*) ii="Full unlimited";DT="60.00";break;;
            *) clear;rm /usr/account/locks/Add.Ip.user;exit 1;;
 esac
dialog --title "Create IP & POP3 Account" --clear \
--yesno "Pay NDS?" 5 40
if [ $? != "0" ]; then
  nds=n
else
  nds=y
fi
 case $nds in
    y*) iii="Pay NDS";break;;
    n*) iii="Not pay NDS";break;;
            *) clear;rm /usr/account/locks/Add.Ip.user;exit 1;;
esac
/usr/account/bin/pasgen >/usr/account/Pass.tmp
c=`cat /usr/account/Pass.tmp`
d=`/usr/account/bin/des $c`
LN=`cat /usr/account/Last.num`
LN=`expr $LN + 1`
dialog --title "Create IP & POP3 Account" --clear --yesno \
"IP login:              ipcc$a
POP3 login:             p3cc$a
domain name:                    $a.elcom.ru
password:               $c
encrypted password:$d
Agreement number: $LN
```

```
Organization name: $on
This client:       $iii
Payment manner:    $ii" 13 70
if [ $? = "0" ]
then
 #dialog --title "Registration in progress... Wait..." --infobox " " 4 60
 echo "Registration in progress... Wait..."
 echo $LN > /usr/account/Last.num
 echo "IP login is: ipcc$a" >/usr/account/Reg.form
 echo "POP3 login is: p3cc$a" >>/usr/account/Reg.form
 echo "domain name is: $a.elcom.ru" >>/usr/account/Reg.form
 echo "IP & POP3 password is: $c" >>/usr/account/Reg.form
 echo "IP & POP3 encoded password is: $d" >>/usr/account/Reg.form
 echo "Agreement number is: $LN" >>/usr/account/Reg.form
 echo "Organization name is: $on" >>/usr/account/Reg.form
 echo "This organization (user) : $iii" >>/usr/account/Reg.form
 echo "Payment manner is: $ii" >>/usr/account/Reg.form
 echo $a >/usr/account/USERLIST.tmp
 cat /usr/account/USERLIST >> /usr/account/USERLIST.tmp
 cp /usr/account/USERLIST.tmp /usr/account/USERLIST
 mkdir /usr/account/users/$a
 case $e in
  o*) break;;
  f*) cp /usr/account/dot/mainctl.cf.fu  /usr/account/users/$a/mainctl.cf;;
  n*) cp /usr/account/dot/mainctl.cf.nu  /usr/account/users/$a/mainctl.cf;;
  c*) cp /usr/account/dot/mainctl.cf.ff  /usr/account/users/$a/mainctl.cf;;
   *) echo "Error...$e";rm /usr/account/locks/Add.Ip.user;exit 1;;
esac
 echo "\$O=\"$on\"" >/usr/account/users/$a/userctl.cf
```

```
        echo "\$N=\"R-$LN\"" >>/usr/account/users/$a/userctl.cf
        echo "\$D=\"$a.elcom.ru\"" >>/usr/account/users/$a/userctl.cf
        echo "\$I=\"ipcc$a\"" >>/usr/account/users/$a/userctl.cf
        if [ x$e = x"c" ]; then
        echo "\$Q=\"No\"" >>/usr/account/users/$a/userctl.cf
        fi
        echo "\$V=\"-\"" >>/usr/account/users/$a/userctl.cf
        if [ $nds = "n" ]; then
         echo "S=0.0 НДС" >>/usr/account/users/$a/userctl.cf
        fi
        echo "END;" >>/usr/account/users/$a/userctl.cf
        date | /usr/account/bin/mkuserdate
        DD=`cat /usr/account/spool/Date.user`
        echo "R-$LN Неполный месяц $DD -$DT" >>
/usr/account/spool/writeoff
        echo "Start remote for DNS"
       /usr/account/bin/acntexec /usr/account/bin/dnsadduser $a
        echo "Start remote for POP3"
       /usr/account/bin/acntexec /usr/account/bin/p3adduser p3cc$a $d $LN
IP_USER_$LN $a.elcom.ru
        echo "Start remote for IP"
       /usr/account/bin/acntexec /usr/account/bin/ipadduser cc$a $d
        echo "Reload DNS..."
       # /usr/account/bin/acntexec /etc/namedb/named.reload
        sleep 5
        mail -s "Important Elcom Rules" postmaster@$a.elcom.ru
</usr/account/templates/Welcome.txt
       # mail -s "Elcom Script (Macros)" postmaster@$a.elcom.ru
</usr/account/templates/Script.txt
       # dialog --title "Create IP & POP3 Account" --clear --msgbox \
```

```
# "Everything is OK! Congratulations !!!!" 5 60
# clear
 rm /usr/account/locks/Add.Ip.user
 exit 0
else
 dialog --title "Create IP & POP3 Account" --clear --msgbox \
 "Try again, my friend !" 5 40
 rm /usr/account/locks/Add.Ip.user
 exit 1
fi
```

3. Практическое задание

1. С помощью команды echo занести в файл text, находящийся в вашем домашнем каталоге, три строки произвольного текста.
2. Создать командный файл, который бы содержал программу на Shell для определения, сколько раз в файле text встречается каждая из букв "a", "e", "o".
3. Поиск каждой буквы оформить в виде функции с параметром.
4. Вывод результатов осуществлять в файл results, расположенный в вашем домашнем каталоге.

4. Вопросы для самоконтроля

1. Что такое Shell?
2. Расскажите о структуре команд.
3. Как осуществляется перенаправление ввода и вывода?
4. Поясните различия между > и >>, < и <<.
5. Какие метасимволы используются в Shell?
6. Как сделать текстовый файл исполнимым?
7. Какие приемы экранирования вы знаете?
8. Расскажите о команде eval.
9. Как осуществляется экспорт переменных?

10. Как осуществляется передача параметров?
11. Сколько параметров можно передать и сколько их будет доступно?
12. Какие подстановки осуществляет Shell-интерпретатор?
13. Что такое программная среда и как получить информацию о стандартных пременных среды? Назовите некоторые из них.
14. Как используются комментарии в Shell?
15. Расскажите о команде test.
16. Перечислите условия проверки файлов.
17. Перечислите условия проверки строк.
18. Перечислите условия сравнения целых чисел.
19. Перечислите сложные условия.
20. Расскажите о структуре условного оператора if.
21. Расскажите о структуре оператора выбора case.
22. Расскажите о структуре оператора цикла с перечислением for.
23. Расскажите о структуре оператора с истинным условием while.
24. Расскажите о структуре оператора с ложным условием until.
25. Расскажите о функциях в Shell.

ЗАКЛЮЧЕНИЕ

Настоящее пособие дает только базовые знания, необходимые для практической работы в качестве администратора ОС UNIX. Для получения более глубоких знаний по взаимодействию параллельных процессов, системному администрированию и программированию в многопользовательских ОС можно обратиться к публикациям, приведенным в списке рекомендуемой литературы, а также к документации и справочникам по ОС UNIX. Тем не менее авторы полагают, что студенты, выполнившие практические работы и изучившие теоретические аспекты, изложенные в пособии, готовы к самостоятельному изучению соответствующих вопросов для дальнейшего повышения квалификации.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. ***Бернстайн Д., Цикритзис Ф.*** Операционные системы. – М.: Мир, 1978.
2. ***Донован Д.*** Системное программирование. – М.: Мир, 1978.
3. ***Каган Б.М.*** Электронные вычислительные машины и системы. - М.: Энергоатомиздат, 1985.
4. ***Королев Л.Н.*** Структуры ЭВМ и их математическое обеспечение. - М.: Наука, 1978.
5. ***Королев М.А., Клемко Г.Н., Мишенин А.И.*** Информационные системы и структуры данных. - М.: Статистика, 1977.
6. ***Монахов М.Ю***. Основы информатики и вычислительной техники. Кн.1. Данные и программы: Учеб. пособие / Под ред. А.В. Кострова; Владим. гос. ун-т. - Владимир,1997.
7. ***Монахов М.Ю***. Основы информатики и вычислительной техники. Кн.2. Информация и арифметика: Учеб. пособие / Под ред. А.В. Кострова; Владим. гос. ун-т. - Владимир, 1998.

8. ***Монахов М.Ю***. Основы информатики и вычислительной техники. Кн.3. Логика и функционирование: Учеб. пособие / Под ред. А.В. Кострова; Владим. гос. ун-т. - Владимир, 1999.
9. ***Монахов М.Ю., Илларионов Ю.А.***. Информатика. Кн.4. Программные и аппаратные части: Учеб. пособие / Владим. гос. ун-т. - Владимир, 2002.
10. ***Мэдник Д., Донован Д.*** Операционные системы. – М.: Мир, 1978.
11. ***Смирнов А.Д.*** Архитектура вычислительных систем: Учеб. пособие для вузов. – М.: Наука, 1990.

Оглавление

Учебное издание

Комплексная защита объектов информатизации

ИЛЛАРИОНОВ Юрий Александрович
ПОЛЯНСКИЙ Дмитрий Александрович
ВВЕДЕНИЕ В СПЕЦИАЛЬНОСТЬ

Практикум

Редактор И.А. Арефьева
Компьютерная верстка Д.А. Полянский

ЛР № 020275. Подписано в печать 10.07.03.
Формат 60x84/16. Бумага для множит. техники. Гарнитура Times.
Печать офсетная. Усл. печ. л. 4,42. Уч-изд. л. 4,62. Тираж 150 экз.
Заказ
Редакционно-издательский комплекс
Владимирского государственного университета.
600000, Владимир, ул. Горького, 87.